Monitoring Aquatic Debris Using Smartphone-Based Robots

Yu Wang, Rui Tan, Guoliang Xing, Jianxun Wang, Xiaobo Tan, Xiaoming Liu, and Xiangmao Chang

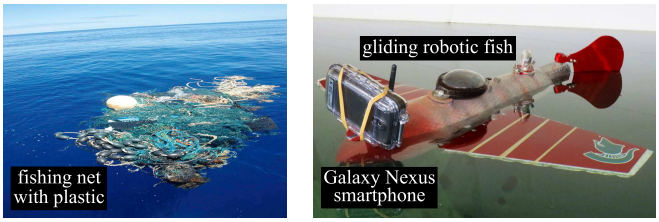
Abstract—Monitoring aquatic debris is of great interest to the ecosystems, marine life, human health, and water transport. This paper presents the design and implementation of SOAR – a vision-based surveillance robot system that integrates an off-the-shelf Android smartphone and a gliding robotic fish for debris monitoring in relatively calm waters. SOAR features real-time debris detection and coverage-based rotation scheduling algorithms. The image processing algorithms for debris detection are specifically designed to address the unique challenges in aquatic environments. The rotation scheduling algorithm provides effective coverage for sporadic debris arrivals despite camera's limited angular view. Moreover, SOAR is able to dynamically offload compute-intensive processing tasks to the cloud for battery power conservation. We have implemented a SOAR prototype and conducted extensive experimental evaluation. The results show that SOAR can accurately detect debris in the presence of various environment and system dynamics, and the rotation scheduling algorithm enables SOAR to capture debris arrivals with reduced energy consumption.

Index Terms—Robotic sensor, aquatic debris, smartphone, computer vision, object detection

1 Introduction

Aquatic debris – human-created waste found in water environments – has emerged to be a grave environmental issue. The 2011 Japan tsunami released about one million tons of debris that heads toward North America [1], and some has drifted to U.S. West Coast as shown in Fig. 1(a). Inland waters also face severe threats from debris. Over 15 scenic lakes in New Jersey still suffer debris resulted from Hurricane Sandy after one year of cleaning [2]. The debris fields pose numerous potential risks to aquatic ecosystems, marine life, human health, and water transport. For example, the debris has led to a loss of up to 4 to 10 million crabs a year in Louisiana [3], and caused damages like propeller entanglement to 58% fishing boats in an Oregon port [4]. It is thus imperative to monitor the debris arrivals and alarm the authorities to take preventive actions for potential risks.

Opportunistic spotting by beach-goers or fishermen is often the only viable solution for small-scale debris monitoring. However, this approach is labor-intensive and unreliable. An alternative approach is *in situ* visual survey by using patrol boats [10]. However, it is costly and can only cover a limited period of time. More advanced methods involve remote sensing technologies, e.g., balloon-board camera [20] and satellite imaging [22]. The former is only effective for one-off and short-term monitoring of highly concentrated debris fields that have been already detected, and the latter often has high operational cost and falls short of monitoring resolution. Recently, autonomous underwater vehicles (AUVs) [15], [32] have been used for various underwater sensing tasks. However, AUV platforms often have high manufacturing costs (over $50,000 per unit [26]). The limitations of these remote sensing and AUV-based approaches make them cost prohibitive for monitoring spatiotemporally scattered debris fields with small-sized objects. For example, the debris from the 2011 Japan tsunami is expected to arrive dispersedly along U.S. West Coast over two years starting from spring of 2012 to late 2014 [1].



(a) (b)

Fig. 1. (a) Debris from the Japan tsunami arriving at U.S. West Coast, 2012 (Photo Credit: Scripps Institution of Oceanography [5]). (b) SOAR prototype integrating a Samsung Galaxy Nexus smartphone in a waterproof enclosure with a gliding robotic fish [6].

This paper presents SOAR (SmartphOne-based Aquatic Robot), a low-cost vision-based robot system that aims to monitor debris in relatively calm waters. It integrates an off-the-shelf smartphone and a robotic fish platform. Fig. 1(b) shows a prototype of SOAR built with a gliding robotic fish developed in our previous work [6] and a Samsung Galaxy Nexus smartphone. Various salient advantages of smartphone and gliding robotic fish make the integration of them a promising platform for debris monitoring. First, recent smartphones are powerful enough to execute advanced computer vision (CV) algorithms to process the images captured by the camera to detect debris objects. Meanwhile, the price of smartphones has been dropping drastically in

• *Yu Wang, Guoliang Xing, and Xiaoming Liu are with the Department of Computer Science and Engineering, Michigan State University, USA.*
• *Rui Tan is with Advanced Digital Sciences Center, Illinois at Singapore, and Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, USA.*
• *Jianxun Wang and Xiaobo Tan are with the Department of Electrical and Computer Engineering, Michigan State University, USA.*
• *Xiangmao Chang is with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China.*

the last five years. Many low-end Android phones (e.g., LG Optimus Net with 800 MHz CPU and 2 GB memory) cost only about $40 [7]. Second, besides visual sensing, various built-in sensing modalities such as GPS and accelerometer can be used to facilitate the navigation and control of the robot and enable situation awareness to improve the debris detection performance. Third, the long-range communication capability of smartphone makes it possible to leverage the cloud to increase robot's intelligence and reduce energy consumption by offloading intensive computation. Lastly, as a commercial off-the-shelf platform, smartphone provides an integrated sensing system and diverse system configurations, which can meet the requirements of a wide spectrum of embedded applications. Moreover, it offers user-friendly programming environments and extensive library support, which greatly accelerates the development process. The gliding robotic fish, which is a low-cost aquatic mobile platform with high maneuverability in rotation and orientation maintenance, provides SOAR the mobility to adapt to the dynamics of debris and water environments. Owing to these features, SOAR represents an unprecedented *vision-based*, *cloud-enabled*, *low-cost*, yet *intelligent* aquatic mobile sensing platform for debris monitoring.

However, the design of SOAR still faces several unique challenges associated with aquatic debris monitoring. First, due to the impact of waves, SOAR cannot acquire a stable camera view, thereby making it highly difficult to reliably recognize the debris objects. A possible solution is image registration [19] that aligns multiple images into a common coordinate system. However, water environments often lack detectable features such as sharp corners that are commonly used for image registration. Second, SOAR is powered by small batteries due to the constraints on the form factor and cost budget, while both aquatic movement of the robot and image processing on the smartphone incur high energy consumption. Lastly, debris arrivals are often sporadic in a large geographic region [10], [13], making them highly difficult to be captured using smartphone cameras that typically have limited field of view. To address these challenges, in this paper we make the following contributions:

1) We develop several lightweight CV algorithms to address the inherent dynamics in aquatic debris detection, which include an image registration algorithm for extracting the horizon line above water and using it to register the images to mitigate the impact of camera shaking, and an adaptive background subtraction algorithm for reliable detection of debris objects.
2) We propose a novel approach to dynamically offloading the compute-intensive CV tasks to the cloud. The offloading decisions are made to minimize the system energy consumption based on *in situ* measurements of wireless link speed and robot acceleration.
3) We analyze the coverage for sporadic and uncertain debris arrivals based on geometric models. Using the analytical debris arriving probability, we design a robot rotation scheduling algorithm that minimizes the movement energy consumption while maintaining a desired level of debris coverage performance.
4) We implement a prototype of SOAR and evaluate it through extensive testbed experiments and trace-driven simulations. The results show that SOAR can accurately detect debris in the presence of various dynamics and maintain a satisfactory level of debris arrival coverage while reducing the energy consumption of robot movement significantly.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 overviews SOAR. Section 4 presents the vision-based debris detection algorithm. Section 5 presents the coverage-based rotation scheduling algorithm. Section 6 discusses several extensions. Section 7 describes the implementation of SOAR. Section 8 presents the evaluation results. Section 9 concludes this paper.

2 Related Work

Current approaches to monitoring aquatic debris fall into three basic categories, including manual spotting, patrol-boat-assisted survey [10], and remote sensing (e.g., balloon-board [20] and satellite imaging [22]). Manual spotting, although viable for small-scale debris monitoring, is label-intensive and lacks robustness. Debris monitoring based on patrol boats and remote sensing is more reliable. However, these approaches are prohibitively expensive for long-term monitoring, especially when debris objects arrive sporadically over vast geographic regions.

Several research efforts have explored the integration of cameras with low-power wireless sensing platforms. Cyclops [24] integrates a CMOS imager hosted by a MICA2 mote [11]. It can perform object detection using a naive background subtraction method. In [32], a low-end camera module is installed on an AUV for navigation. However, these camera-based platforms can only conduct simple image processing tasks due to the resource constraints of motes. Recently, mobile sensing based on smartphones has received increasing research interest due to their rich computation, communication, and storage resources. The study in [34] designs a driving safety alert system that can detect dangerous driving behaviors using both front- and rear-facing cameras of a smartphone. This paper aims to design an aquatic debris surveillance robot that utilizes the built-in camera, inertial sensors, and other resources on smartphone. Different from existing vision-based systems, we need to deal with unique challenges in aquatic debris monitoring, such as camera shaking and sporadic debris arrivals.

Extracting the foreground objects from a sequence of video frames is a fundamental CV task. Background subtraction [19] is a widely adopted approach, which, however, often incurs significant computation overhead to resource-constrained devices. In [27], compressive sensing is applied for background subtraction to reduce computation overhead. In [36], an adaptive background model is proposed to trade off the object detection performance and computation overhead of background subtraction. These approaches assume a static camera view, and hence cannot be readily applied to the debris detection in water environments where camera is constantly shaking due to waves. This paper develops a collection of vision-based detection algorithms that are specifically designed for background subtraction in dynamic water environments and optimized for smartphone platforms.

Several approaches have been proposed to maintain monitoring coverage with cameras. In [14], the placement of static cameras is determined according to a floor plan. A camera network deployment approach is developed in [17] to minimize the coverage overlap between neighboring cameras. Different from these approaches that focus on static cameras, we exploit the controllable mobility of robot to increase the likelihood of capturing debris objects. Moreover, unlike previous studies (e.g., [9]) that use mobile cameras to cover fixed locations of interest, this paper aims to reduce the miss rate in covering sporadic debris arrivals.

3 Overview of SOAR

SOAR consists of an off-the-shelf Android smartphone and a gliding robotic fish. The smartphone is loaded with an app that implements the CV, movement scheduling, and cloud communication algorithms. The gliding robotic fish is capable of moving in water by beating its tail that is driven by a servo motor. The motor is manipulated by a programmable control board, which can communicate with the smartphone through either a USB cable or short-range wireless links such as ZigBee. Various closed-loop motion control algorithms based on smartphone's built-in inertial sensor readings can be implemented on either fish control board or smartphone.

SOAR is designed to operate on the surface of relatively calm waters, i.e., with mild waves like ripples, and monitor floating debris in nearshore aquatic environments such as public recreational beaches where wireless (cellular/WiFi) coverage is available. We focus on monitoring static or slow-moving on-water objects, and filter out other objects such as boats and swimmers based on the estimated speed. When a long shoreline needs to be monitored, multiple SOAR nodes can be deployed dispersedly to form barrier coverage. In this case, the number of needed nodes is the ratio of the length of the monitored shoreline to the coverage range of the smartphone's built-in camera. In this paper, we focus on the design of debris detection and mobility scheduling algorithms running on a single SOAR node. The sensing results of multiple nodes can be sent back to a central server via the long-range communication interface on smartphones for fusion and human inspection. SOAR has a limited sensing area due to the angular view of the built-in camera on smartphone[1], which makes it difficult to capture the debris arrivals that are likely sporadic [10], [13]. Mobility can be exploited to address this challenge. The gliding robotic fish is capable of both rotating and moving forward. As a rotation can be achieved by beating the fish tail once, it consumes much less energy than moving forward that requires continuous tail beats. Thus, this paper exploits the rotation mobility of the robotic fish[2] and assumes that the SOAR remains relatively stationary in water. In still water or slow water current, feedback motion control can maintain SOAR's station. In the presence of fast water current, an anchor can be used together with motion control to reduce energy consumption in maintaining a stationary position.

After the initial deployment, SOAR begins a debris surveillance process consisting of multiple *monitoring round*s. In each round, SOAR executes a rotation schedule, which includes the camera orientation and an associated monitoring time interval. Specifically, SOAR rotates to the desired orientation at the beginning of a round and starts to take images at a certain rate, which is determined by a sleep/wake duty cycle. For each image, SOAR uses several CV algorithms to detect the existence of debris objects in real time. Between two image captures, SOAR sleeps to save energy. At the end of a round, SOAR computes the rotation schedule for the next round based on the detection results to ensure a desired level of debris coverage. SOAR is designed to achieve long-term (up to a few months) autonomous debris monitoring. In addition to duty cycling, it adopts a novel offloading approach to leveraging the cloud for battery power conservation. Specifically, SOAR comprises the following two major information processing components.

Real-time debris detection: This component aims to extract debris objects from the taken images. It consists of three lightweight image processing modules, i.e., *image registration*, *background subtraction*, and *debris identification*, which can effectively deal with various environment and system dynamics such as shaking and internal noise of the camera. Specifically, SOAR first registers each frame by exploiting the unique features in aquatic environments, e.g., the horizon line and shoreline. Then, background subtraction in HSV color space is performed on the registered frame to identify the foreground pixel candidates. Finally, the foreground is passed to debris identification for noise removal and debris recognition. At runtime, SOAR minimizes the energy consumption by determining if the above image processing tasks should be locally executed or entirely/partially offloaded to the cloud depending on the current network condition, e.g., the wireless network availability and link speed.

Coverage-based rotation scheduling: On the completion of a monitoring round, SOAR analyzes the debris coverage performance based on the estimated debris movement orientation and the surveillance history. It then adaptively configures the camera orientation and monitoring time interval for the next round. Because of the limited energy supply and power-consuming movement in water environments, SOAR must efficiently adjust its orientation while maintaining a desired level of debris coverage performance. To this end, we propose a scheduling algorithm that minimizes the rotation energy consumption in a round by dynamically configuring the rotation schedule, subject to a specified upper bound on miss coverage rate for debris arrivals.

4 Real-Time Debris Detection

The image processing pipeline of SOAR is illustrated in Fig. 2. Although it is based on a collection of elementary

1. Extra optical components like fisheye lens can be used to broaden the camera view. However, the integration of these components will complicate the design of SOAR. In particular, the image processing algorithms will incur significantly higher computation overhead due to the distortion in captured images (see Section 8.1.1).

2. Extra mechanical components like rotor can be used to rotate the camera only. However, this will complicate the system design and may negatively affect the system reliability. For example, it will bring uncertainty in weight balancing and water proofing of the gliding robotic fish [6].

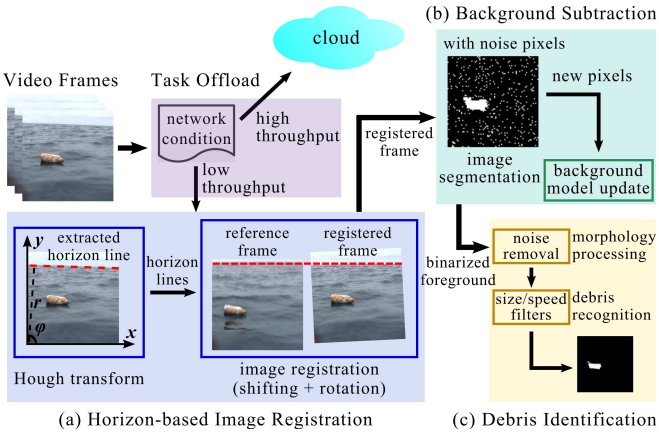


Fig. 2. Real-time debris objects detection.

CV algorithms, it is non-trivial to optimize these compute-intensive synergistic algorithms for smartphones given the limited resources and stringent requirement on system lifetime. Specifically, SOAR consists of the following image processing components. The *image registration* aligns consecutive frames to mitigate the impact of camera shaking caused by waves. In this paper, we focus on aquatic environments with observable horizon line as an example, although our techniques can adapt to other image features. The horizon line can be used as a reference to register the frames. To deal with the high computation overhead of horizon line extraction, SOAR offloads a portion of computation to the cloud based on the network link speed and shaking levels indicated by the smartphone's inertial sensor readings. The registered frames are then compared with a background model to extract the foreground. Lastly, the *debris identification* removes the salt-and-pepper noises from foreground image and identifies the debris objects.

4.1 Horizon-based Image Registration

Image registration is the process of aligning images taken at different time instants into one coordinate system [19]. In debris detection, image registration is necessary to mitigate the impact of camera shaking caused by waves, such that subsequent pixel-wise processing can be executed properly. Registration is performed by establishing correspondence between images based on their distinguishable features. However, a key challenge is that there are few detectable image features in typical water environments that can guide the image registration. A novelty of our approach is to leverage the horizon line, which segments the sky and water areas, for image registration, as shown in Fig. 2(a).

We employ Hough transform [18] to extract the horizon line. Hough transform has been widely used to identify the positions of lines in an image. We assume that the majority of an image is either sky or water area, which are separated by the horizon line. As the sky and water areas typically have distinct colors [13], Hough transform is able to extract the horizon line accurately. For each frame, we first convert it to a grayscale image and detect edges using a Sobel operator [19]. The Sobel operator detects an edge point according to the local intensity of gradient magnitude. Hough transform then finds the horizon line through a voting process based on the number of edge points that each candidate line passes through. The result of Hough transform is a line expressed as $r = x \cdot \cos\phi + y \cdot \sin\phi$ in the Cartesian coordinate system originated at the bottom-left corner of an image. The horizon line is parameterized by r and ϕ, which are the distance from the horizon line to the origin and the angle between the horizon line's orthogonal direction and x-axis, respectively. An illustration of the extracted horizon line is shown in Fig. 2(a).

Based on the extracted horizon line, we register each video frame to mitigate the impact of camera shaking. Specifically, for two consecutive frames, we register the successor frame according to the registered predecessor by aligning their extracted horizon lines. Let Δy denote the vertical shifting at the midpoint of the horizon lines, and η denote the angle that the horizon line rotates in these two frames. We assume that the midpoints of the horizon lines in the two frames correspond to the same point in the world plane. Such an assumption is motivated by the observation that the closed-loop motion control algorithms can maintain the robot's orientation and position by adaptive course-correction. Therefore, Δy and η define the *affine transform* between these two consecutive frames. First, we shift the successor frame to align the midpoint of its horizon line with that of the registered predecessor frame by $x' = x$ and $y' = y + \Delta y$, where (x, y) are the coordinates of a pixel in the unregistered frame, and (x', y') are the corresponding coordinates of (x, y) after shifting. Then, we rotate the frame by

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} \cos\eta & \sin\eta & x_0(1-\cos\eta) - y_0\sin\eta \\ -\sin\eta & \cos\eta & x_0\sin\eta - y_0(1-\cos\eta) \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix},$$

where (x_r, y_r) denote the coordinates of the underlying pixel in the reference frame, and (x_0, y_0) denote the coordinates of horizon line midpoint in the successor frame after shifting and serve as the center of rotation. For pixels without their correspondents in the original unregistered frame due to the rotation, we adopt bilinear interpolation to fill the color data (i.e., RGB) for them.

4.2 Background Subtraction

To reduce the energy consumption in image processing, we adopt a lightweight background subtraction approach to detecting the foreground debris objects. We first convert the representation of an image to HSV (Hue, Saturation, and Value) model. In HSV, hue represents the color, saturation is the dominance of hue, and value indicates the lightness of the color. The HSV representation is robust to illumination changes and hence more effective in interpreting color features in the presence of reflection in water environments.

In our approach, the background of a pixel is represented by a Gaussian mixture model (GMM) [19]. The GMM comprises K three-dimensional Gaussians, where each Gaussian characterizes the three channels of HSV color space. When a new frame is available, each pixel is compared with its background model. If the HSV vector of a pixel in the new frame does not fall within a certain range from any mean vector of the K Gaussians, this pixel is considered a foreground pixel candidate; otherwise, it is classified as

background. Therefore, the color difference between the foreground and background affects the classification accuracy. In our implementation, the range is chosen to be 2.5 times of the standard deviation of the corresponding Gaussian. Under this setting, the image segmentation can tolerate minor inaccuracy introduced by noises and accommodate certain environmental condition changes.

Any classified pixel will be used to update the GMM. In GMM, the k^{th} ($k \in \{1, 2, \ldots, K\}$) Gaussian has a weight (denoted by $c_{k,i}$), which characterizes the fraction of pixels till frame i that support it. If none of the existing K Gaussians match the new pixel in frame $i+1$ (i.e., this pixel has been classified as foreground), the distribution with the lowest weight is replaced by a new Gaussian, which is assigned with a large variance, a small weight, and a mean vector equal to this new pixel; otherwise, the weight, mean vector, and variance of the matched Gaussian(s) are updated based on the new pixel value and a learning rate γ. Specifically, the k^{th} Gaussian $\mathcal{N} \sim (\mathbf{m}_k, \Sigma_k)$ with a matched background pixel $\mathbf{x}_{i+1} \in \mathbb{R}^3$ is updated as follows:

$$\begin{aligned} c_{k,i+1} &= (1-\gamma)\cdot c_{k,i} + \gamma, \\ \mathbf{m}_{k,i+1} &= (1-\rho)\cdot \mathbf{m}_{k,i} + \rho \cdot \mathbf{x}_{i+1}, \\ \Sigma_{k,i+1} &= (1-\rho)\cdot \Sigma_{k,i} + \rho \cdot (\mathbf{x}_{i+1} - \mathbf{m}_{k,i})^{\top} \cdot (\mathbf{x}_{i+1} + \mathbf{m}_{k,i}), \end{aligned} \tag{1}$$

where $\rho = \gamma \cdot \exp(-\frac{1}{2}(\mathbf{x}_{k,i+1} - \mathbf{m}_{k,i})^{\top} \Sigma_{k,i}^{-1} (\mathbf{x}_{k,i+1} - \mathbf{m}_{k,i})) \cdot (2\pi)^{-\frac{3}{2}} \cdot |\Sigma_{k,i}|^{-\frac{1}{2}}$. We note that a larger learning rate γ represents weaker relevance of historical observations and thus is suitable for scenarios with rapidly changing background. The above update scheme allows the GMM to adapt to environmental condition changes, enhancing the robustness of background subtraction. Moreover, it can reduce the false alarms caused by dirt or water on the camera as the GMM is updated with the affected frames.

We now discuss the impact of the number of Gaussians (i.e., K) in the GMM on background substraction performance. As shown in [30], the background model with $K = 1$ can only deal with static background. Therefore, more Gaussians are needed in aquatic debris detection to account for the dynamics from camera shaking, reflection, and noises. A larger K can enrich the information maintained by the GMM and hence improve its robustness. However, it also imposes additional computation overhead for the smartphone. In Section 8.1.5, we will evaluate the trade-off between system overhead and detection performance through experiments, which guides the setting of K. We note that it may require a large K to describe the environments with more complex and dynamic background. Our approach can be easily extended to employ existing online algorithms that maintain a GMM with variable K adaptive to background changes, such as the reversible jump Markov chain Monte Carlo method [25] and its variant [31] that has been used for video processing.

4.3 Debris Identification

The binarized foreground image contains randomly distributed noise pixels, as depicted in Fig. 2(b). Because the background subtraction is conducted in a pixel-wise manner, the labeling of foreground and background can be affected by camera noise, resulting in false foreground pixels. To deal with these noise pixels, we adopt the opening operation in image morphology [19]. The opening operation, which consists of erosion followed by dilation, eliminates the noise pixels through erosion while preserving the true foreground by dilation. After the noise removal, we employ region growing to identify the debris objects from the foreground image. It uses the foreground pixels as the initial seed points and forms connected regions that represent the candidate debris objects by merging nearby foreground pixels.

We note that the extracted foreground candidate objects may contain objects that can move actively, e.g., boats and swimmers. SOAR adopts a threshold-based approach to filter out these non-debris objects. Specifically, the robot estimates the object movement speed based on the pinhole camera projection model (see Section 5.4). When the estimated speed is higher than a threshold, SOAR will save the image and periodically transmit back to a central server for human inspection. The threshold on speed is chosen to exclude the non-debris objects with active mobility. Similar heuristics based on object moving orientations can be applied to improve the accuracy of debris recognition. We use another threshold-based method to remove the small objects. The small object size in the image usually indicates a false alarm or a distant debris object. Ignoring them does not affect the system accuracy since distant real debris objects will likely be detected when they approach closer to the camera. Note that the shape of debris object has little impact on the debris detection performance, as the false negatives in detection are mainly caused by the high color similarity between foreground and background and the long distance between object and camera.

4.4 Dynamic Task Offloading

A key advantage of smartphone-based robots lies in their capability of leveraging the abundant resources of the cloud. To prolong the smartphone battery lifetime, SOAR dynamically offloads the entire/partial image processing to the cloud when there is network coverage. As the typical coverage of a cellular tower is up to several miles [29], cellular networks can be available in nearshore water surface. The offloading decision mainly depends on two factors: 1) the overhead of image processing algorithms, e.g., the power consumption, when they are executed locally; and 2) the wireless network condition, e.g., the uploading speed, which determines the energy consumption for uploading images to the cloud. In our design, SOAR has three offloading schemes, i.e., *cloud*, *local*, and *hybrid* processing, and dynamically chooses a scheme with the lowest energy consumption. The energy consumption of these schemes are analyzed as follows.

4.4.1 Cloud and Local Processing Schemes

We first analyze the energy consumption of the *local* and *cloud* processing schemes, where all the processing on the original frame is conducted either in the cloud or on the phone. When a new frame is available, SOAR checks the network (cellular/WiFi) link speed and estimates the delay to upload this frame. The energy consumption for uploading the entire image is given by $e_{\text{cloud}} = p_c(\gamma) \cdot s/\gamma$, where

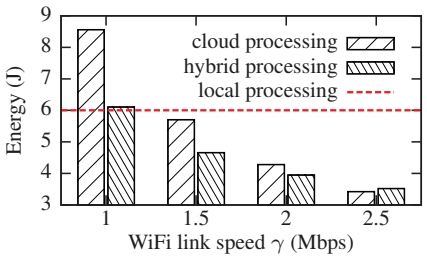


Fig. 3. Energy consumption per frame on Samsung Galaxy Nexus.

s is the frame size, γ is the measured link speed, and $p_c(\gamma)$ is the uploading power consumption under the link speed γ. Alternatively, the frame can be processed on the phone. Let t_l denote the delay to process a frame locally, including *image registration*, *background subtraction*, and *debris identification*. Our measurements show that these modules have similar CPU power consumption, which is denoted by p_l. The energy drain for processing a frame on the phone is thus given by $e_{\text{local}} = p_l \cdot t_l$.

4.4.2 Hybrid Processing Scheme

In addition to the above two options, we propose a hybrid solution that offloads the Hough transform in *image registration* to the cloud and conducts the rest of processing locally. Such a design is motivated by the observation that the Hough transform incurs nearly 70% processing overhead for a frame (see Section 8.1.1). Moreover, as the energy consumption for offloading is largely determined by the uploading volume, we propose to upload a rectangular part of the original frame, which contains the horizon line. As the camera is shaking, the selection is based on the horizon line in the reference frame and the accelerometer readings. Specifically, we adopt the accumulated linear vertical acceleration (denoted by $\sum a_z$) over the time duration from the predecessor reference frame to the current frame to quantify the camera shaking. In theory, if $\sum a_z < 0$, the horizon line will shift upward in the current frame; otherwise, it will shift downward. Therefore, from the sign of $\sum a_z$, we can estimate whether the horizon line in the current frame is in the upper or lower part divided by the horizon line in the reference frame. We verify this hypothesis using 20 video sequences with each consisting of 30 frames. The results show that this hypothesis holds for 576 frames out of all 600 frames. In our hybrid scheme, the rectangular part to be uploaded has the original frame width and a height (denoted by h_c) from the horizon line midpoint in the unregistered reference frame to either width depending on the sign of $\sum a_z$. Let h_0 denote the original frame height, and t_h denote the delay to conduct Hough transform locally. The energy consumption for hybrid processing is given by $e_{\text{hybrid}} = \frac{h_c}{h_0} e_{\text{cloud}} + \frac{t_l - t_h}{t_l} e_{\text{local}}$. Note that this formula ignores the low-probability cases where the above hypothesis does not hold. In these cases, the cloud will fail to identify the horizon line and the original frame will be processed locally.

4.4.3 Minimize Energy Consumption

By comparing e_{local}, e_{hybrid}, and e_{cloud}, SOAR chooses a scheme with the lowest energy consumption. All the parameters except the height h_c in hybrid scheme and the link speed γ can be obtained using offline measurements. In our current prototype, we use WiFi to upload video frames to the cloud, although the implementation can be easily extended to cellular networks (3G/4G). We measure the power consumption of a Samsung Galaxy Nexus using an Agilent 34411A multimeter when the smartphone is uploading video frames under various WiFi link speed settings and locally processing the frames. Fig. 3 plots the energy consumption per frame under the three schemes. Note that for the hybrid scheme, we set $h_c/h_0 = 50\%$. We can observe that when the link speed is high, it is preferable to offload the entire/partial image processing to the cloud for energy conservation. Note that our dynamic offloading scheme does not rely on a particular type of wireless network. If cellular networks are used, the trend shown in Fig. 3 will also hold. The only difference is that the offloading decision will be made at a link speed higher than that in Fig. 3, because cellular interface consumes more energy per bit transmission than WiFi [12].

5 Coverage-Based Rotation Scheduling

In this section, we first introduce camera sensing and debris arrival models, and analyze the effectiveness of covering debris arrivals. We then present a rotation scheduling algorithm that aims to minimize the rotation energy consumption while maintaining a desired coverage rate for debris arrivals.

5.1 Camera and Debris Models

The *field of view* (FOV) is a widely used concept to characterize a camera's sensing area, in which any object with dimensions larger than pre-defined thresholds can be reliably detected from the taken images. A camera's FOV is typically represented by a sector originated at the camera's pinhole with an angular view α and a radius R [19], where α is hardware-dependent and R can be measured through experiments for a specific application. For example, in pedestrian detection, the camera's FOV has a radius of up to 40 meters [16]. For the objects within the FOV, image sensing is insensitive to their dimensions as long as they are larger than a certain size. For example, our on-water experiments show that any floating object with a cross-sectional area over $0.28\,\text{m}^2$ (e.g., a five gallon water bottle) can be reliably detected 40 meters away from a smartphone's camera. We note that FOV is a conservative approximation to the camera's sensing area because the objects outside FOV may also be detected by the camera. This approximation simplifies the analysis of the coverage of debris arrivals. In Section 5.2, we will show that the rotation scheduling algorithm does not depend on the value of R.

Since we focus on the nearshore debris arrival monitoring, the *surveillance region* for SOAR is defined as the semi-circular area originated at the robot with a radius of R. We define the camera orientation by its optical axis. Because of the limited angular coverage of FOV, SOAR needs to constantly adjust its orientation to maintain a desired coverage level for debris arrivals over the defined surveillance region. In this paper, we assume that the adjustments of camera orientation only occur at time instants that are multiples of a fixed time interval unit, which is referred to as *slot*.

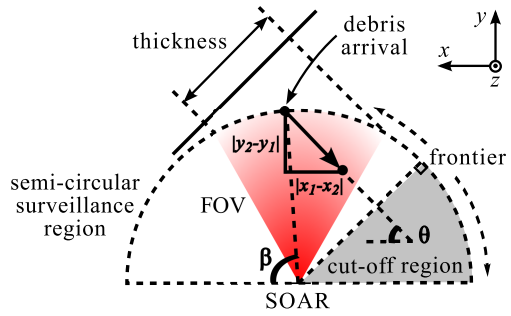


Fig. 4. Illustration of FOV, surveillance region, thickness, debris arriving angle β, debris movement orientation θ and its estimation, and the cut-off region.

In the nearshore water environments, debris objects often drift passively with water current that moves straightly in a fixed direction [13]. We set up a Cartesian coordinate system with x-axis, y-axis, and z-axis, parallel, horizontally perpendicular, and vertically perpendicular to the shoreline, respectively. We assume that the positions of debris objects on the water surface at a given time instant follow the Poisson point process, which has been verified in previous studies [10], [13]. Our analysis (omitted due to space limit) shows that the number of debris objects that arrive at the surveillance region *frontier*, as illustrated in Fig. 4, follows the Poisson process. A Poisson process is characterized by an *arrival rate*, denoted by λ, and a *time interval*, denoted by τ. The λ is the expected number of arriving debris objects at the surveillance region during a slot, and τ is the number of slots. The probability that there is at least one object arriving at the surveillance region during interval τ is

$$P_a = 1 - \mathrm{e}^{-\lambda\tau}. \tag{2}$$

As the occurrences of debris objects are rare events, the λ is small. In Section 5.4, we will describe an approach to estimating λ based on the historical detection results and envision a cloud-based approach that leverages the publicly available information about aquatic debris.

5.2 Debris Arrival Coverage

In the rotation scheduling, an arriving debris object is *covered* by SOAR if the arrival position at the surveillance region frontier falls into the camera's FOV. A rotation schedule aims to cover as many newly arriving debris objects as possible. Accordingly, our rotation scheduling algorithm controls the orientation of SOAR based on the statistics of the historical debris arrival observations. Therefore, it is important to model the arrivals of debris objects at the surveillance region. We note that whether a debris object covered by the camera's FOV can be eventually extracted from the taken images is determined by the performance of the CV algorithms (see Section 4). However, it is desirable to maximize the coverage rate, which improves the overall debris monitoring performance. We define the coverage rate for debris arrivals provided by SOAR as the ratio of the number of trajectories hitting camera's FOV frontier to that hitting surveillance region frontier. However, one immediate challenge is that lines are uncountable. We employ *thickness* in geometric probability [28] to measure the set of trajectories. Specifically, the thickness of an arc frontier, denoted by $\mathcal{T}$, is defined as the length of its projection to the line perpendicular to debris movement orientation, as illustrated in Fig. 4.

The thickness allows us to directly evaluate the debris coverage performance. We first analyze the probability of *miss coverage* based on thickness. Let θ denote the *debris movement orientation*, which is the angle between the debris' trajectory and x-axis. Let β denote the *debris arriving angle*, which is the angle between the radius from debris' arrival position at the surveillance region frontier and x-axis. Both θ and β are illustrated in Fig. 4. To simplify the discussion, we discretize the possible arriving angles by 1°, and define *arrival frontier* as the 1° arc centered at the arrival position. Let $\mathcal{T}(\beta)$ and $\mathcal{T}_0$ denote the thickness of arrival frontier and the thickness of the whole surveillance region frontier, respectively. Based on the geometric relationship, we have $\mathcal{T}(\beta) = 2R\sin 0.5^\circ \cos(\beta-\theta)$ and $\mathcal{T}_0 = (1+\sin\theta)R$. Therefore, the probability that a debris object hits the arrival frontier defined by β conditioned that its trajectory intersects with the surveillance region frontier, is given by $g(\beta,\theta) = \mathcal{T}(\beta)/\mathcal{T}_0$, which is independent of R. As the probability that at least one debris object arrives at the surveillance region is P_a given in Eq. (2), the probability that at least one debris object reaches the arrival frontier defined by β (denoted by $P_a(\beta)$) is $P_a(\beta) = P_a \cdot \mathcal{T}(\beta)/\mathcal{T}_0 = P_a \cdot g(\beta,\theta)$. We note that if β is not covered by the FOV during an interval τ, $P_a(\beta)$ represents the probability of missing debris arrivals. We then derive the miss coverage rate, which will be used to schedule the rotation of SOAR. Let $t_r(\beta)$ denote the end time instant of the most recent slot when β was covered by the camera's FOV, and t denote the end time instant of interval $[t_r(\beta), t]$, during which the arrival frontier defined by β remains uncovered. The *miss coverage rate* at arriving angle β and time t, denoted by $\omega(\beta, t \,|\, t_r(\beta), \theta)$, is defined as the probability of missing debris arrivals during interval $[t_r(\beta), t]$. Formally,

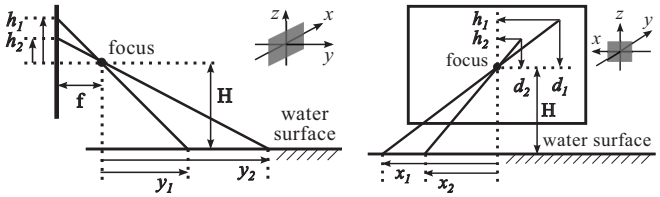
$$\begin{aligned}\omega(\beta, t \,|\, t_r(\beta), \theta) &= P_a(\beta)\,|_{\,\tau = t - t_r(\beta)} \\ &= (1 - \exp(-\lambda(t - t_r(\beta)))) \cdot g(\beta,\theta).\end{aligned} \tag{3}$$

Note that the miss coverage rate does not depend on the specific value of R, but the ratio of thicknesses, i.e., $g(\beta,\theta)$. Thus, the coverage-based rotation scheduling algorithm presented in Section 5.3 will not rely on R.

5.3 Coverage-based Rotation Scheduling

Because of the limited angular coverage and energy budget, the rotation of SOAR needs to be carefully scheduled to achieve the desired coverage rate for debris arrivals. In this section, we formulate the problem of coverage-based robot rotation scheduling. Our objective is to schedule the next camera orientation and corresponding monitoring interval to minimize the overall energy consumption, subject to an upper-bounded miss coverage rate at all arrival angles.

As the rotation energy consumption is approximately proportional to the orientation change [21], we adopt the changed orientation to characterize the consumed energy in rotation. Let $\widetilde{\theta}$ denote the estimated debris movement orientation (see Section 5.4). Suppose the current camera orientation and time instant are β_0' and t'. The next rotation



(a) Pinhole projection y-z. (b) Pinhole projection x-z.

Fig. 5. Pinhole camera projection model.

schedule, including the camera orientation β_0 and end time instant of monitoring interval t, minimizes the average rotation rate $\overline{v}$:

$$\overline{v} = \frac{|\beta_0 - \beta_0'|}{t - t'}, \tag{4}$$

subject to

$$\omega(\beta, t \,|\, t_r(\beta), \widetilde{\theta}) < \xi, \ \forall\, \beta \in [0, \beta_0 - \alpha/2] \cup [\beta_0 + \alpha/2, \pi], \tag{5}$$

where ξ is a user-specified maximum tolerable miss coverage rate. The constraint in Eq. (5) upper-bounds the miss coverage rate ω at each uncovered β. Note that for β within the camera's FOV, i.e., $\beta \in [\beta_0 - \alpha/2, \beta_0 + \alpha/2]$, it has zero miss coverage rate. The above problem can be efficiently solved by searching a set of discrete candidate orientations. During surveillance, SOAR keeps a map of ω at each β, and updates it before each new scheduling to account for system dynamics, which includes the error in orientation adjustment, the updated $\widetilde{\theta}$ (see Section 5.4) and λ. With the updated map of ω, SOAR adaptively schedules the next orientation and the associated monitoring interval.

Eq. (5) ensures an upper bound on ω for uncovered arriving angles. Given the debris movement orientation θ, there always exists a *cut-off region* as illustrated in Fig. 4. Specifically, the arrival position of a debris object will never fall into the frontier of this cut-off region. To avoid the unnecessary rotation, the surveillance region can exclude this cut-off region. In Section 8.2.5, we will evaluate the reduction in energy consumption by truncating the surveillance region.

5.4 Debris Movement and Arrival Estimation

From Eq. (3), the miss coverage rate ω depends on the debris movement orientation θ. Before deployment, SOAR can be configured with a coarsely estimated θ from prior knowledge about the water movement direction in the deployment region. Once SOAR detects a debris object, θ can be accurately estimated based on the pinhole camera projection model and the positions of the object in images. Fig. 5 shows the pinhole projection model. Specifically, Fig. 5(a) illustrates how real-world distance along y-axis is projected to vertical axis h in the image. It can be obtained that $|y_1 - y_2| = fH \cdot |1/h_2 - 1/h_1|$, where y_1 and y_2 are distances between SOAR and the debris object in two frames; h_1 and h_2 are the vertical pixel distance equivalents of y_1 and y_2; f is the camera focal length; and H is the mounting height of smartphone on SOAR. Similarly, Fig. 5(b) illustrates how real-world distance along x-axis corresponds to horizontal axis d. We have $|x_1 - x_2| = H \cdot |d_2/h_2 - d_1/h_1|$, where x_1, x_2, d_1, and d_2 are similar measures to y_1, y_2, h_1, and h_2. Based on the geometric relation shown in Fig. 4, the estimated debris movement orientation $\widetilde{\theta}$ is given by

$$\widetilde{\theta} = \arctan \left| \frac{y_1 - y_2}{x_1 - x_2} \right| = \arctan \left(f \cdot \left| \frac{h_1 - h_2}{d_2 h_1 - d_1 h_2} \right| \right) + \theta_r, \tag{6}$$

where θ_r is the heading direction of SOAR and can be obtained from the built-in digital compass of smartphone. We will evaluate the accuracy of $\widetilde{\theta}$ in Section 8.1.3. Moreover, the object movement speed can be estimated by $(|x_1 - x_2|^2 + |y_1 - y_2|^2)^{1/2} / |t_1 - t_2|$, where t_1 and t_2 are the time instants when the object is at $(x_1, y_1, 0)$ and $(x_2, y_2, 0)$, respectively.

We then describe two approaches to estimating the debris arrival rate λ. First, λ can be estimated based on the historical detection results by SOAR. Suppose the robot detects n debris objects in the rotation schedule specified by β_0 and t. As discussed in Section 5.2, the probability that an arriving debris object is covered by the camera's FOV conditioned that it arrives at the surveillance region is given by the ratio of their thicknesses, i.e., $\mathcal{T}_{\text{FOV}}/\mathcal{T}_0$. Hence, the expectation of debris arrivals at the whole surveillance region during one slot, i.e., λ, can be estimated by $n\mathcal{T}_0/(P_d \mathcal{T}_{\text{FOV}}(t - t'))$, where P_d is the lower bound on detection probability for the FOV and $t - t'$ counts the monitoring slots. Second, the long-range communication capability of SOAR makes it possible to exploit the available web resources to estimate λ. For example, the Marine Debris Clearinghouse [8] is a representative online tool for aquatic debris tracking. Based on satellite images, on-site reports, and aquatic field simulations, it can estimate the intensity of incoming debris over large areas.

6 Discussions

6.1 Barrier Coverage with Multiple SOARs

In this section, we extend our approach to monitoring a long shoreline with multiple SOAR nodes. SOAR nodes can be deployed along the monitored shoreline to form a barrier while performing the vision-based debris detection and coverage-based rotation scheduling without inter-node coordinations. By adjusting the deployment density, we can balance the debris coverage performance and system cost in terms of the number of SOARs. We characterize the deployment density and barrier coverage performance using the distance between two adjacent SOARs (denoted by d) and average miss coverage rate (denoted by $\overline{\omega}$), respectively. We first consider the sparse case where SOARs are dispersedly deployed so that their surveillance regions do not overlap while the debris arrivals at the monitored shoreline can be fully covered. Given the cut-off region (see Section 5.3), we have $d_{\max} = R(1 + 1/\sin\theta)$ where θ is the debris movement orientation. This deployment minimizes the number of SOARs subject to $\overline{\omega} < \xi$ where ξ is the maximum tolerable miss coverage rate for a single SOAR. On the other hand, SOARs can be deployed densely so that they can cover the arriving debris objects without rotation. Therefore, SOARs maintain FOVs with thicknesses making up that of the monitored shoreline. Based on the geometric relationship, we have $d_{\min} = R(1 - \cos\alpha + \sin\alpha \cot\theta)$ where α is the angular view of FOV. Under this deployment, SOARs can achieve zero miss coverage rate, i.e., $\overline{\omega} = 0$, at the cost

of increased number of nodes. Fig. 6(a) shows the impact of deployment density when SOARs (with $\alpha = 5\pi/18$) are used to monitor debris arrivals (with $\theta = \pi/3$) at a $50R$-long shoreline. In summary, adapting the deployment density of SOARs allows us to trade off the overall system cost and debris coverage performance.

6.2 Adaptability to Harsh Environments

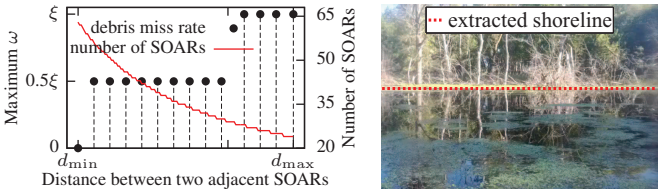
We now discuss the adaptability of SOAR to more harsh environments.

Alternative image features. In previous sections, we focus on debris detection in aquatic environments with observable horizon line and use it as the reference to register the captured images. In waterways such as inland lakes and rivers where the horizon line is unavailable, the shoreline that segments the water and non-water areas can be leveraged to guide the image registration. In this case, our debris detection algorithm can be directly applied since the Hough transform identifies the shoreline in the same manner as extracting the horizon line (see Section 4.1). To evaluate the image registration performance, we deploy our prototype in an inland lake [33]. Fig. 6(b) shows a typical frame captured by SOAR and the extracted shoreline. As we can see, although the frame has fairly complex background, our approach can accurately detect the shoreline. When these line-based features are absent, our approach can be extended to employ other more complex image features such as specific geometric shapes. Specifically, the Hough transform detects the target feature via a voting process in the corresponding parameter space. For example, a circle-based feature is determined in a three-dimensional (i.e., radius and center coordinates) parameter space. Once the target feature is extracted, the rest of image processing still applies. We note that our debris detection algorithm will incur higher computation overhead in this case, as the complexity of Hough transform increases with the number of parameters. We will evaluate this in Section 8.1.1.

Strong waves. This paper focuses on debris detection in relatively calm waters, i.e., with mild waves like ripples. We note that, in the presence of strong waves, the image features (e.g., the horizon line and shoreline) may be out of the camera view due to shaking. In this case, SOAR can skip the current frame. Alternatively, our debris detection algorithm can be extended to employ existing image registration schemes that do not rely on observable features, such as the inertia-based image registration [33] that leverages the built-in inertial sensors of smartphone. We will evaluate the impact of camera shaking caused by mild waves in Section 8.1.6.

7 System Prototype

We have built a proof-of-concept prototype of SOAR for evaluation. The vision-based debris detection algorithm presented in Section 4 is fully implemented on smartphone platforms running Android 4.3 Jelly Bean. System evaluation is mainly conducted on two representative handsets, a Samsung Galaxy Nexus (referred to as *Galaxy*) and a Google Nexus 4 (referred to as *Nexus4*). The implementation takes about 1.99 MB storage on the phone, and requires about 10.2 MB RAM when the frame resolution is set to be 720×480. When a new frame is available, SOAR checks the current WiFi condition using Android API `WifiInfo.getLinkSpeed()`. Based on the measured link speed and horizon line position, it determines whether this newly arrived frame is locally processed or entirely/partially uploaded to the cloud, following the scheme proposed in Section 4.4. Our initial implementation of image processing modules in Java incurs extensive delays on current Android system. To boost the frame processing performance, we use OpenCV libraries and interface them with Java using Java Native Interface on Android. In particular, we adopt OpenCV's native implementation of the Hough transform, which is more efficient than other implementations according to our tests.



(a) (b)

Fig. 6. (a) Impact of deployment density on barrier coverage. (b) Shoreline detection in an inland lake.

We integrate the smartphone to a gliding robotic fish developed in our previous work [6]. The fish platform weights 9 kilogram and represents a hybrid of underwater gliders and robotic fish with advantages of both. It is equipped with a ZigBee radio for wireless communication, two $75\,\mathrm{W}\cdot\mathrm{h}$ on-board batteries, and a circuit board for mobility control, localization, and navigation. On the control board, we implement a closed-loop proportional-integral-derivative (PID) controller that adaptively configures the tail beat based on the discrepancy between the scheduled and actual orientations. In our current implementation, we use a host computer to relay the communication between the smartphone (using WiFi) and the fish (using ZigBee). In the future, we will establish direct connection between them, and migrate the PID controller to the smartphone to reduce the physical size and cost of the fish control board. Fig. 1(b) depicts our prototype system that integrates a Galaxy in a water-proof enclosure with a gliding robotic fish.

8 Performance Evaluation

We evaluate SOAR through testbed experiments and simulations based on data traces collected from the prototype. The testbed experiments validate the feasibility of SOAR by evaluating the computation overhead, effectiveness of each module, and the overall performance of a fully integrated SOAR. The simulations extensively evaluate the performance of SOAR working under wide ranges for parameter settings.

8.1 Testbed Experiments

We evaluate SOAR in controlled lab experiments to fully understand its performance. We conduct extensive experiments using our prototype in a 15×10 feet water tank in

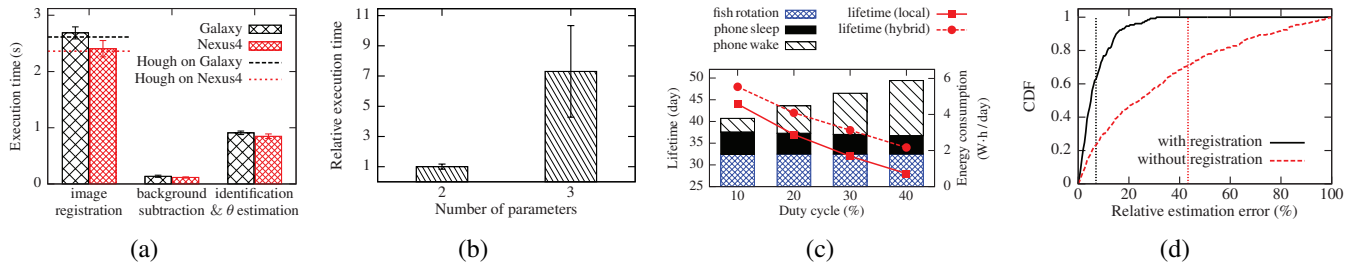


Fig. 7. (a) Execution time of image processing modules on phones. (b) Complexity of Hough transform vs. number of parameters. (c) Projected lifetime and daily energy consumption. (d) CDF and average of the relative estimation error for θ.

our lab. Along one side of the tank, we vertically place a piece of white foam board above the water surface to imitate the sky area. The line where the foam board intersects with the water surface produces an artificial horizon line. In the experiments, we set the camera frame rate to be 0.25 fps. The setting is based on the fact that debris usually has a slow drifting speed and hence does not require a rapidly updated background model. Moreover, a low frame rate helps prolong the battery lifetime. The GMM comprises 3 three-dimensional Gaussians (i.e., $K = 3$), unless otherwise specified. We test the debris detection performance under various environmental conditions and experimental settings, which include different camera shaking levels, with and without registration, and different settings of GMM. For each scenario, we conduct nine runs of experiments. For each run, we calculate the detection probability as the ratio of frames with correct detections to the total frames with debris, and the false alarm rate as the ratio of frames with false detections to the total frames without debris.

8.1.1 Overhead on Smartphone Hardware

We first evaluate the overhead of the vision-based detection algorithms on smartphone platforms. Specifically, we measure the computation delay of each image processing module, i.e., *image registration*, *background subtraction*, and *debris identification* on Galaxy and Nexus4, respectively. Galaxy has an 1.2 GHz dual-core processor and 1 GB memory, and Nexus4 has an 1.5 GHz quad-core processor and 2 GB memory. They are representative mid- and high-end mainstream smartphone platforms. The computation delay is measured as the elapsed time using Android API `System.nanoTime()`. The results are plotted in Fig. 7(a). We can see that *background subtraction* takes the least time, followed by *debris identification* combined with debris movement orientation estimation. *Image registration* incurs the longest delay. Breakdown shows that this long delay is mostly caused by the Hough transform. The overall computation delay is within 3.7 and 3.3 seconds on Galaxy and Nexus4, respectively, which well meet the real-time requirement of debris monitoring as debris arrivals are typically sporadic [10], [13].

We then evaluate the impact of the number of parameters on the complexity of Hough transform. The Hough transform will need more than 2 parameters to characterize the target image feature if it is either not line-based or captured by a fisheye camera with distorted view [19]. In this set of experiments, we collect 337 images of various sizes, where each image has features both in line (modeled by 2 parameters) and circle (modeled by 3 parameters). For each image, we record the computation delay of performing the Hough transform to detect the line-based and circle-based features, respectively. Fig. 7(b) plots the measured computation delay of Hough transform versus the number of parameters. As we can see, if the target feature needs to be modeled by 3 parameters, the computation delay of Hough transform is about 7 times of that of detecting a line-based feature. Therefore, the complexity of Hough transform drastically increases with the number of parameters.

TABLE 1
Power consumption profile of SOAR.

	voltage (V)	current (A)	power (W)
phone wake	3.7	0.439	1.624
phone sleep	3.7	0.014	0.518
servo motor	6	0.5	3

8.1.2 Projected Lifetime

This set of experiments evaluate the lifetime of SOAR based on its power consumption profile shown in Table 1. SOAR has a total battery capacity of $170\,\mathrm{W\cdot h}$, including a backup $13.5\,\mathrm{W\cdot h}$ and two main $75\,\mathrm{W\cdot h}$ batteries on the fish, and a $6.48\,\mathrm{W\cdot h}$ battery on the smartphone. Its major energy consumption is due to the smartphone during wake periods and the fish rotation. According to our results from the integrated evaluation (see Fig. 10(b)), a monitoring round lasts for 5 minutes averagely, and SOAR can rotate to the scheduled orientation within 15 seconds. We can thus calculate the upper bound on daily (12 hours of daytime) energy consumption for fish rotation as $(12{\times}60/5){\times}(15/3600){\times}p_r\,\mathrm{W\cdot h}$, where p_r is the power consumption for fish rotation. The energy drain on the smartphone can be calculated using the offline power consumption measurements (denoted by p_s and p_w) and duty cycle. In particular, the power consumption during wake periods (i.e., p_w) is calculated as the average of power consumption measurements when the smartphone is running the *hybrid* processing scheme. This mode involves frame capture, processing, and communication. Therefore, p_w is an upper bound for the average power consumption during wake periods. We acknowledge that these offline measurements provide a simplified analysis of smartphone power consumption and they ignore the dynamics in power states such as the tail effect mentioned by the reviewer. Our problem formulation can be improved by integrating fine-grained power models that are based on online measurements of component-level power consumption [35].

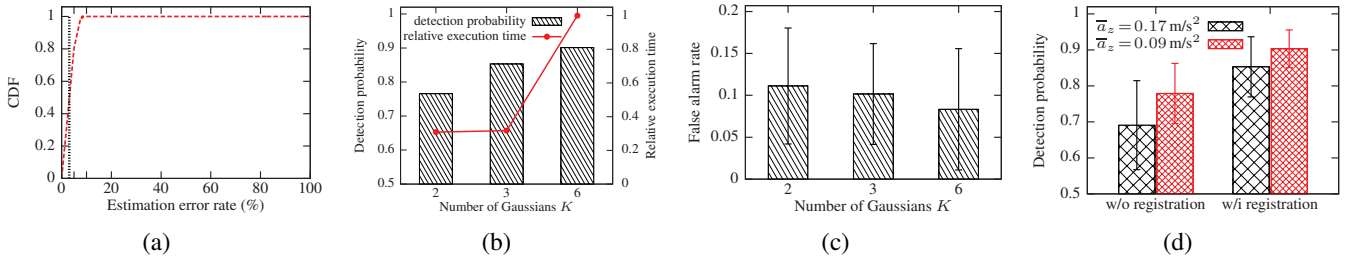


Fig. 8. (a) CDF and average of horizon line location estimation. (b) Impact of the number of mixed Gaussians K in GMM. (c) False alarm rate vs. the number of mixed Gaussians K. (d) Impact of image registration and shaking level.

Fig. 7(c) plots the projected lifetime under various duty cycle settings, when all the CV tasks are conducted on the smartphone. The duty cycle is defined as the ratio of wake time to the total time. As expected, the lifetime decreases with duty cycle. However, a low duty cycle will decrease the temporal sensing granularity of SOAR. For example, during a 5 minutes monitoring interval, it can capture and process about 15 and 37 frames under 20% and 50% duty cycles, respectively, with a frame rate of 0.25 fps. The breakdown of SOAR daily energy consumption is also shown in Fig. 7(c). We find that the majority of energy is consumed by the wake periods and fish rotation. For example, when running the *local* processing scheme with a duty cycle of 40%, SOAR consumes $3.05\,\mathrm{W\cdot h}$, $1.02\,\mathrm{W\cdot h}$, and $1.80\,\mathrm{W\cdot h}$ energy for wake periods, sleep periods, and fish rotation, respectively. Thus, the daily energy consumption is around $5.87\,\mathrm{W\cdot h}$. Note that the daily energy consumption of a smartphone can be around $1\,\mathrm{W\cdot h}$ when it keeps in the sleep mode [23]. Moreover, Fig. 7(c) shows the projected lifetime when smartphone runs the *hybrid* scheme under a link speed of 2 Mbps. The *hybrid* scheme reduces the power consumption of the smartphone during wake periods by offloading the intensive Hough transform to the cloud. It can be seen that the *hybrid* scheme leads to 9.1% to 21.5% improvement on lifetime under different duty cycles.

8.1.3 Accuracy of Debris Movement Orientation Estimation

We then evaluate the debris movement orientation estimation presented in Section 5.4. Initially, the debris movement orientation θ is unknown to SOAR. After a debris object is successfully detected in two frames, SOAR can estimate θ based on Eq. (6). In the experiments, an object (a can) is fastened to a rope. We drag the object using the rope to simulate the movement of debris object. We define the *relative estimation error* as $|\theta-\widetilde{\theta}|/\theta$, where $\widetilde{\theta}$ is the estimated orientation from Eq. (6) and θ is the ground truth obtained by using a protractor. Fig. 7(d) plots the cumulative distribution function (CDF) and average of the relative estimation error for the approaches with and without image registration. We can see that the approach with registration can accurately estimate θ, with an average relative estimation error of only about 7%. In contrast, the approach without registration results in an average relative estimation error of around 43%.

8.1.4 Accuracy of Horizon Line Location Estimation

In this set of experiments, we evaluate the accuracy of horizon line location estimation used in the dynamic task offloading. Specifically, the hybrid processing scheme estimates the rough location of the horizon line in terms of shifting direction based on the measured acceleration, thus avoiding the compute-intensive Hough transform. We define the *estimation error rate* as the ratio of frames with incorrect estimation of shifting direction to the total frames in this run of experiment. Fig. 8(a) shows the CDF and average of the estimation error rate. We can see that our approach accurately estimates the shifting direction of the horizon line, achieving an average estimation error rate of about 3%. We note that, in the case with incorrect estimation, the cloud will notify SOAR of the failure and request the CV tasks to be locally executed.

8.1.5 Impact of the Number of Mixed Gaussians

We now evaluate the impact of the number of mixed Gaussians (i.e., K) on detection performance. Section 4.2 discusses the trade-off between the detection performance and system overhead caused by the setting of K. The detection probability and execution time on Galaxy are plotted in Fig. 8(b). We can see that the detection probability increases with K. Moreover, Fig. 8(c) evaluates the false alarm rate versus K, where the error bar represents the standard deviation. It can be observed that the false alarm rate decreases with K. A larger K imposes higher computation overhead in both image segmentation and background model update. When the GMM adopts 2 Gaussians, the computation delay is about 30% of that with 6 Gaussians. From the figures, we also find that the setting $K=3$ achieves a satisfactory trade-off between detection performance and computation delay. Therefore, we set $K=3$ in other testbed experiments.

8.1.6 Effectiveness of Image Registration

As discussed in Section 4.2, the background subtraction is conducted in a pixel-wise manner, hence its performance is sensitive to camera shaking. Fig. 9 shows a sample of background subtraction. Specifically, Fig. 9(a) is the original frame where the red and black dashed lines represent the extracted horizon lines for this frame and the registered predecessor frame, respectively. Fig. 9(b) shows the background model, where each pixel is the mean vector of the Gaussian with the largest weight in the GMM. Fig. 9(c) is the result of background subtraction without image registration. Fig. 9(d) is the result with image registration before subtraction. We can see that our horizon-based registration effectively mitigates the impact of camera shaking, and hence the detection algorithm can more accurately pinpoint the foreground object location in the image. Fig. 8(d) plots

Fig. 9. Sample background subtraction outputs for the approaches with/without horizon-based image registration: (a) the original frame, (b) Gaussian component with the largest weight, (c) detection result without image registration, (d) detection result with image registration.

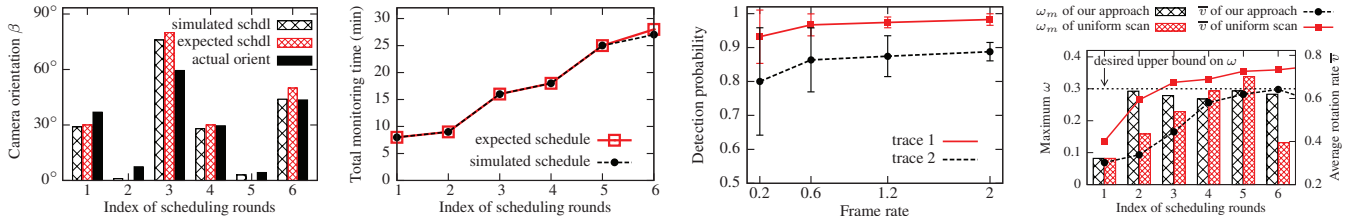


Fig. 10. (a) Simulated, expected, and actual monitoring orientations. (b) Simulated and actual monitoring intervals. (c) Impact of frame rate on debris detection probability. (d) Maximum ω and average $\overline{v}$ vs. index of scheduling rounds.

the detection probability for approaches with and without image registration under different camera shaking levels. In the experiments, we generate different levels of waves by controlling a feed pump connected to the tank. The $\overline{a}_z$ reported in Fig. 8(d) is the average linear vertical acceleration (i.e., excluding gravity) measured by the built-in accelerometer on smartphone, and hence characterizes the camera shaking levels. We can see that the image registration not only improves the average detection performance, but also decreases the variance in detection probability in the presence of camera shaking. It effectively mitigates the impact of shaking, leading to a smaller degradation in detection probability as camera shaking increases.

8.1.7 *Integrated Evaluation*

In this set of experiments, all modules of SOAR (vision-based debris detection, θ estimation, rotation scheduling, and PID-controlled orientation adjustment) are integrated and evaluated. Similar to previous experiments, we drag a can to simulate a debris object. The debris movement orientation θ is 0.156π, which is unknown to the system before deployment. We set the slot duration as 1 minute. At time $t = 0$, SOAR is deployed perpendicularly to the tank length. At time $t = 1$ min, it starts the first monitoring round as discussed in Section 3. In this experiment, SOAR achieves 83.3% detection probability and 5.8% average relative θ estimation error. We further study the performance of rotation scheduling and orientation adjustment of SOAR. It is unlikely for SOAR to rotate to the exact scheduled orientation due to complex fluid dynamics and compass inaccuracy. This orientation adjustment error and minor inaccuracy in θ estimation affect the rotation scheduling for the next round. For evaluation, we compare the actual rotations of SOAR with the real-time scheduled rotations during this experiment (referred to as *expected schedule*), and those in an ideal simulation (referred to as *simulated schedule*) where we assume both orientation adjustment and θ estimation are accurate and thus feed the scheduling algorithm with ground truth. We note that the differences between the actual rotations and expected schedule characterize the performance of the PID-controlled orientation adjustment. The deviations between the expected schedule and simulated schedule indicate the robustness of the rotation scheduling algorithm to the control and estimation errors. The results are shown in Fig. 10(a) and Fig. 10(b). We find that the orientation adjustment errors vary for different expected orientations due to different levels of fluid obstruction. For example, in the 3$^{\text{rd}}$ scheduling round, SOAR is subject to a higher level of fluid obstruction when it targets an expected orientation perpendicular to the water movement direction. Our PID controller generally maintains an orientation adjustment error lower than $15°$. Moreover, as shown in Fig. 10(b), the expected monitoring intervals well follow the simulated schedule because the temporal scheduling is mainly determined by the debris arriving intensity over time.

8.2 Trace-driven Simulations

To more extensively evaluate SOAR under realistic settings, we conduct simulations driven by real data traces collected using our prototype in the water tank. The data include error traces of SOAR orientation adjustments and video traces of debris arrivals. First, the error traces of SOAR orientation adjustments are collected using our prototype system. The camera orientation is represented by the heading direction of SOAR. We collect the error traces by measuring the discrepancy between the desired orientation and actual heading direction. Second, we use our prototype to collect two distinct sets of video traces of an arriving bottle. In our prototype, the phone is mounted about 3 inches above the water surface. Trace 1 is collected in calm environment, and has 1495 frames in total. Trace 2, with a total of 1275 frames,

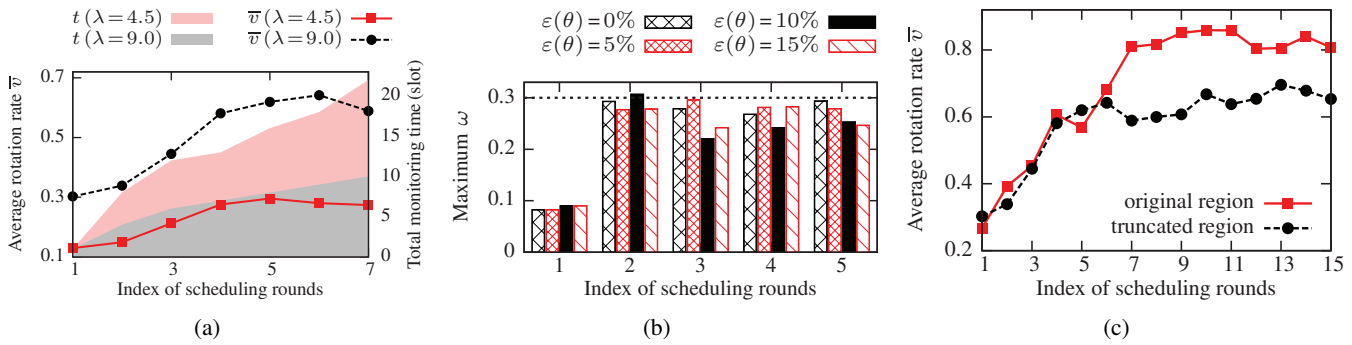


Fig. 11. (a) Impact of arrival rate on rotation rate and frequency. (b) Impact of estimation error on debris movement orientation θ. (c) Impact of surveillance region truncation on rotation rate.

is collected in the presence of persistent waves generated by the feed pump. To provide ground truth, the foreground debris object in each frame is manually labeled.

In the simulations, SOAR is deployed to monitor debris objects arriving in a semi-circular surveillance region. The arrival rate λ is set to be 9, unless otherwise specified. The debris movement orientation is $\theta = \pi/3$, and we assume that θ is known to the robot. We set the FOV angular coverage α to be $5\pi/18$ based on our measurements of Galaxy. Initially, the robot is deployed perpendicular to the shoreline. It is allowed to adjust its orientation after the first slot. For each orientation adjustment, the actual direction is set according to the collected traces, which is thus subject to discrepancies from the desired orientation.

8.2.1 Impact of Frame Rate on Detection Performance

Fig. 10(c) plots the detection probability versus frame rate. We can observe that the detection probability increases with frame rate. This is because a higher frame rate enables the GMM to be more timely updated to capture the environmental condition changes. However, the improvement gained by increasing frame rate is fairly limited. The reasons are two-fold. First, debris objects usually drift slowly with water current. The GMM can thus be updated with a low rate. Second, our horizon-based image registration effectively mitigates the impact of camera shaking, which is the major affecting factor for debris detection. Hence, a low frame rate can achieve satisfactory debris detection performance. Moreover, in terms of system overhead, a low frame rate is desirable for smartphone platforms that have constraints on resources and energy supply.

8.2.2 Coverage Effectiveness

We compare the coverage effectiveness of our approach with a heuristic baseline approach that uniformly scans the surveillance region. Specifically, the baseline approach evenly partitions the semi-circular surveillance region into $\lceil \pi/\alpha \rceil$ sub-regions. In this scheme, the robot sequentially scans the sub-regions by making an orientation adjustment each slot. For our approach, the desired upper bound on miss coverage rate is set to be 0.3. We evaluate the coverage effectiveness by examining the maximum miss coverage rate (denoted by ω_m) among all arriving angles. The ω_m characterizes the worst-case debris coverage performance for a rotation schedule. Fig. 10(d) plots the ω_m versus index of scheduling rounds. We can observe that our approach can guarantee the upper bound on miss coverage rate, since it adaptively allocates surveillance slots based on the ω at each β, while the uniform scan cannot bound the ω_m. Moreover, we evaluate the average rotation rate $\overline{v}$, where a larger $\overline{v}$ indicates higher power consumption. Fig. 10(d) also plots the $\overline{v}$ versus index of scheduling rounds. As the uniform scan approach continuously adjusts the orientation, it consumes more power than our approach.

8.2.3 Impact of Arrival Rate

Fig. 11(a) plots the average rotation rate $\overline{v}$ versus index of scheduling rounds under different settings of debris arrival rate λ. We can see that the robot has a higher $\overline{v}$ when λ is higher. This is consistent with the intuition that the robot needs to rotate faster when debris arrives more frequently. Fig. 11(a) also plots the total monitoring time versus index of scheduling rounds. We can see that for a higher λ, the rotation scheduling has to be conducted more frequently to meet the upper-bounded miss coverage rate ω, as ω at the uncovered arriving angles increases with λ. The robot is scheduled to rotate less frequently under a lower λ, resulting in a lower $\overline{v}$ and a longer monitoring interval at each orientation. Overall, the results in Fig. 11(a) demonstrate that our scheduling algorithm can adaptively control the robot rotation to achieve the desired coverage performance while minimizing the energy consumption.

8.2.4 Impact of Estimation Errors

This set of simulations evaluate the impact of estimation errors of debris movement orientation θ on debris coverage performance. Let $\varepsilon(\cdot)$ denote the relative estimation error with respect to ground truth. Fig. 11(b) plots the maximum miss coverage rate ω_m versus index of scheduling rounds under various estimation error levels for θ. We note that the estimation error in θ affects the scheduling of camera orientation, as θ determines the distribution of debris arriving probability at the frontier of surveillance region. From the figure, we can see that our rotation scheduling algorithm generally maintains ω_m below the desired upper bound of 0.3, as long as the relative estimation errors for θ is below 15%. As shown in Section 8.1, $\varepsilon(\theta)$ is only about 7%. Thus, SOAR can tolerate practical inaccuracy in estimating θ. Moreover, our analysis (omitted due to space limit) validates that SOAR shows similar tolerance to the inaccuracy in estimating debris arrival rate λ.

8.2.5 Impact of Surveillance Region Truncation

Finally, we evaluate the surveillance region truncation scheme presented in Section 5.3. Given the debris movement orientation $\theta = \pi/3$, there exists a cut-off region $[5\pi/6, \pi]$ where the debris arrival positions will never fall into its frontier. Hence, the surveillance region can exclude this cut-off region to avoid unnecessary robot rotation. Fig. 11(c) plots the average rotation rate $\overline{v}$ for the approaches with full and truncated surveillance regions, respectively. From the figure, we can observe that the $\overline{v}$ is reduced by about 25% after excluding the cut-off region. Therefore, the proposed surveillance region truncation can effectively improve the energy efficiency of SOAR without sacrificing the debris coverage performance.

9 Conclusion and Future Work

This paper presents SOAR – a new vision-based robotic sensor system designed for aquatic debris monitoring. SOAR integrates an off-the-shelf Android smartphone and a gliding robotic fish. The vision-based debris detection algorithm of SOAR effectively deals with various dynamics such as camera shaking and reflection. A rotation scheduling algorithm adaptively guides the rotation of SOAR to capture the images of arriving debris objects. Moreover, SOAR dynamically offloads the entire/partial image processing to the cloud for energy conservation. Testbed experiments and extensive simulations based on a prototype system show that SOAR provides robust debris detection performance, meets the real-time requirement on smartphone platforms, and efficiently covers the sporadic debris arrivals.

In our future work, we plan to deploy SOAR in an inland lake and evaluate it under various conditions like debris flow speed and brightness/lightening. Moreover, we will develop multi-SOAR coordination schemes, including the fusion of images taken by different robots and collaborative movement/rotation scheduling algorithms for increased spatiotemporal coverage. Lastly, we will update the robotic fish platform of SOAR to provide more flexibility in integrating with smartphone, e.g., rotating the smartphone via a rotor, to reduce the rotation energy consumption.

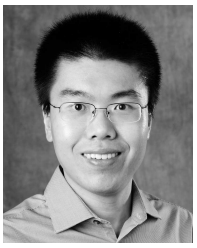
Acknowledgments

This work was supported in part by the U.S. National Science Foundation under grants CNS-0954039 (CAREER), ECCS-1446793, CCF-1331852, CNS-1218475, CNS-1059373, IIP-1343413, IIS-1319602, IIS-0916720, and the National Nature Science Foundation of China under grant No. 61202350.

References

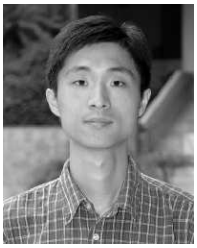
[1] [Online] http://marinedebris.noaa.gov/tsunamidebris, 2015.
[2] [Online] http://usat.ly/LWoxTI, 2013.
[3] [Online] http://water.epa.gov/type/oceb/marinedebris, 2015.
[4] [Online] http://waterboards.ca.gov/water_issues, 2015.
[5] [Online] http://projectkaisei.org, 2012.
[6] [Online] http://nbcnews.to/1fGAomj, 2013.
[7] [Online] http://amzn.to/Y4BvO9, 2015.
[8] [Online] http://clearinghouse.marinedebris.noaa.gov, 2015.
[9] J. Ai and A. Abouzeid, "Coverage by directional sensors in randomly deployed wireless sensor networks," *J. Combinatorial Optimization*, vol. 11, no. 1, pp. 21–41, 2006.
[10] R. Boland and M. Donohue, "Marine debris accumulation in the nearshore marine habitat of the endangered Hawaiian monk seal," *Marine Pollution Bull.*, vol. 46, no. 11, pp. 1385–1394, 2003.
[11] Crossbow Technology, MICA2, TelosB datasheets, 2015.
[12] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," in *The 8th Int. Conf. Mobile Systems, Applications, and Services (MobiSys)*, 2010, pp. 49–62.
[13] J. Davies, J. Baxter, M. Bradley, D. Connor, J. Khan, E. Murray, W. Sanderson, C. Turnbull, and M. Vincent, *Marine Monitoring Handbook*. Joint Nature Conservation Committee, 2001.
[14] U. Erdem and S. Sclaroff, "Automated camera layout to satisfy task-specific and floor plan-specific coverage requirements," *Computer Vision and Image Understanding*, vol. 103, no. 3, pp. 156–169, 2006.
[15] C. Eriksen, J. Osse, R. Light, T. Wen, T. Lehman, P. Sabin, J. Ballard, and A. Chiodi, "Seaglider: a long-range autonomous underwater vehicle for oceanographic research," *IEEE J. Ocean. Eng.*, vol. 26, no. 4, pp. 424–436, 2001.
[16] T. Gandhi and M. Trivedi, "Pedestrian protection systems: issues, survey, and challenges," *IEEE Trans. Intell. Transp. Syst.*, vol. 8, no. 3, pp. 413–430, 2007.
[17] M. Hoffmann, M. Wittke, J. Hahner, and C. Muller-Schloer, "Spatial partitioning in self-organizing smart camera systems," *IEEE J. Sel. Topics Signal Process.*, vol. 2, no. 4, pp. 480–492, 2008.
[18] J. Illingworth and J. Kittler, "A survey of the Hough transform," *Computer Vision, Graphics, and Image Processing*, vol. 44, no. 1, pp. 87–116, 1988.
[19] A. Jain, *Fundamentals of Digital Image Processing*. Prentice Hall, 1989, vol. 3.
[20] S. Kako, A. Isobe, and S. Magome, "Low altitude remote-sensing method to monitor marine and beach litter of various colors using a balloon equipped with a digital camera," *Marine Pollution Bull.*, vol. 64, no. 6, pp. 1156–1162, 2012.
[21] N. Leonard and J. Graver, "Model-based feedback control of autonomous underwater gliders," *IEEE J. Ocean. Eng.*, vol. 26, no. 4, pp. 633–645, 2001.
[22] T. Mace, "At-sea detection of marine debris: overview of technologies, processes, issues, and options," *Marine Pollution Bull.*, vol. 65, no. 1, pp. 23–27, 2012.
[23] B. Priyantha, D. Lymberopoulos, and J. Liu, "Littlerock: Enabling energy-efficient continuous sensing on mobile phones," *Pervasive Computing*, vol. 10, no. 2, pp. 12–15, 2011.
[24] M. Rahimi, R. Baer, O. Iroezi, J. Garcia, J. Warrior, D. Estrin, and M. Srivastava, "Cyclops: in situ image sensing and interpretation in wireless sensor networks," in *The 3rd Int. Conf. Embedded Networked Sensor Systems (SenSys)*, 2005, pp. 192–204.
[25] S. Richardson and P. Green, "On Bayesian analysis of mixtures with an unknown number of components," *J. Royal Statistical Soc.*, vol. 59, no. 4, pp. 731–792, 1997.
[26] D. Rudnick, R. Davis, C. Eriksen, D. Fratantoni, and M. Perry, "Underwater gliders for ocean research," *J. Marine Technology Soc.*, vol. 38, no. 2, pp. 73–84, 2004.
[27] Y. Shen, W. Hu, J. Liu, M. Yang, B. Wei, and C. Tungchou, "Efficient background subtraction for real-time tracking in embedded camera network," in *The 10th Int. Conf. Embedded Networked Sensor Systems (SenSys)*, 2012, pp. 295–308.
[28] H. Solomon, *Geometric Probability*. Soc. for Industrial and Applied Math., 1978, vol. 28.
[29] C. Song, Z. Qu, N. Blumm, and A.-L. Barabási, "Limits of predictability in human mobility," *Science*, vol. 327, no. 5968, pp. 1018–1021, 2010.
[30] C. Stauffer and E. Grimson, "Adaptive background mixture models for real-time tracking," in *The 13th IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 1999, pp. 246–252.
[31] R. Tan, H. Huo, J. Qian, and T. Fang, "Traffic video segmentation using adaptive-k Gaussian mixture model," in *Advances in Machine Vision, Image Processing, and Pattern Analysis*, 2006, pp. 125–134.
[32] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke, "Data collection, storage, and retrieval with an underwater sensor network," in *The 3rd Int. Conf. Embedded Networked Sensor Systems (SenSys)*, 2005, pp. 154–165.
[33] Y. Wang, R. Tan, G. Xing, J. Wang, X. Tan, and X. Liu, "Samba: a smartphone-based robot system for energy-efficient aquatic environment monitoring," in *The 14th ACM/IEEE Int. Conf. on Information Processing in Sensor Networks (IPSN)*, 2015.

[34] C. You, N. Lane, F. Chen, R. Wang, Z. Chen, T. Bao, Y. Cheng, M. Lin, L. Torresani, and A. Campbell, "CarSafe app: alerting drowsy and distracted drivers using dual cameras on smartphones," in *The 11th Int. Conf. Mobile Systems, Applications, and Services (MobiSys)*, 2013, pp. 13–26.
[35] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *The 8th Int. Conf. Hardware/Software Codesign and System Synthesis*, 2010, pp. 105–114.
[36] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," in *The 17th Int. Conf. Pattern Recognition (ICPR)*, 2004, pp. 28–31.



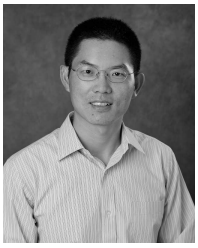
Yu Wang is a Senior Software Engineer at Samsung Electronics America, Bellevue, USA. He received the Bachelor's degree in electrical engineering from Nanjing University, Nanjing, China, in 2010, and the Ph.D. degree in computer science from Michigan State University, East Lansing, USA, in 2015. His research interests include signal/information processing, mobile computing, and cyber-physical systems.

Rui Tan is a Research Scientist at Advanced Digital Sciences Center, a Singapore-based research center of the University of Illinois at Urbana-Champaign (UIUC). He is also jointly appointed by Coordinated Science Laboratory of UIUC. He received the B.S. and M.S. degrees in automation from Shanghai Jiao Tong University, Shanghai, China, in 2004 and 2007, respectively, and the Ph.D. degree in computer science from City University of Hong Kong, Hong Kong SAR, in 2010. He worked as a Research Associate at Michigan State University, East Lansing, USA, from 2010 to 2012. His research interests include sensor networks, cyber-physical systems, and security.

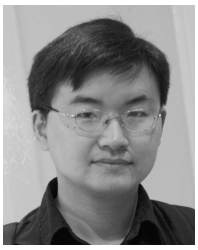


Guoliang Xing is an Associate Professor with the Department of Computer Science and Engineering at Michigan State University, East Lansing, USA. He received the B.S. degree in electrical engineering and M.S. degree in computer science from Xi'an Jiao Tong University, China, in 1998 and 2001, respectively, and the M.S. and D.Sc. degrees in computer science and engineering from Washington University in St. Louis, in 2003 and 2006, respectively. From 2006 to 2008, he was an Assistant Professor of computer science with City University of Hong Kong, Hong Kong SAR. His research interests include sensor networks, mobile systems, and cyber-physical systems. Dr. Xing was an NSF CAREER Award recipient in 2010. He received the Best Paper Awards at the 18th IEEE Int. Conf. Network Protocols (ICNP) in 2010, and the 12th ACM/IEEE Int. Conf. Information Processing in Sensor Networks (IPSN) in 2013, respectively.

Jianxun Wang is a Mechantronic Engineer at Apple, Cupertino, USA. He received the B.S. degree in automation from Harbin Institute of Technology, Harbin, China, in 2009, and the Ph.D. degree in electrical and computer engineering from Michigan State University, East Lansing, USA, in 2014. His research interests include robotics and human-machine interactions.



Xiaobo Tan is a Professor with the Department of Electrical and Computer Engineering and the Department of Mechanical Engineering (by courtesy) at Michigan State University, East Lansing, USA. He received the B.E. and M.E. degrees in automatic control from Tsinghua University, Beijing, China, in 1995 and 1998, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Maryland, College Park, USA, in 2002. His research interests include bio-inspired underwater robots and their application to environmental sensing, electroactive polymer sensors and actuators, and modeling and control of systems with hysteresis. Dr. Tan has (co)authored one book (*Biomimetic Robotic Artificial Muscles*) and over 60 journal papers, and holds one U.S. patent with two more pending. He was a recipient of NSF CAREER Award (2006), MSU Teacher-Scholar Award (2010), and several Best Paper Awards.



Xiaoming Liu is an Assistant Professor with the Department of Computer Science and Engineering at Michigan State University (MSU), East Lansing, USA. He received the B.E. degree in computer science from the Beijing Information Technology Institute, Beijing, China, in 1997, the M.E. degree in computer science from Zhejiang University, Hangzhou, China, in 2000, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, USA, in 2004. Before joining MSU in Fall 2012, he was a Research Scientist with the General Electric Global Research Center, Niskayuna, USA. His research areas include face recognition, biometrics, image alignment, video surveillance, computer vision, and pattern recognition. Dr. Liu has authored over 80 scientific publications, and holds 22 U.S. patents.

Xiangmao Chang is an Associate Professor at Nanjing University of Aeronautics and Astronautics, Nanjing, China. He received the B.S. degree in mathematics from Liao Cheng University, Liaocheng, China, in 2004, the M.S. degree in mathematics from Beijing Jiaotong University, Beijing, China, in 2007, and the Ph.D. degree in computer science from Beijing University of Posts and Telecommunications, Beijing, China, in 2011. His research interests include sensor networks and cyber-physical systems.