The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

Real-time Moving Object Recognition and Tracking Using Computation Offloading

Yamini Nimmagadda, Karthik Kumar, Yung-Hsiang Lu, and C. S. George Lee

Abstract— **Mobile robots are widely used for computation-intensive tasks such as surveillance, moving object recognition and tracking. Existing studies perform the computation entirely on robot processors or on dedicated servers. The robot processors are limited by their computation capability; real-time performance may not be achieved. Even though servers can perform tasks faster, the communication time between robots and servers is affected by variations in wireless bandwidths. In this paper, we present a system for real-time moving object recognition and tracking using *computation offloading*. Offloading migrates computation to servers to reduce the computation time on the robots. However, the migration consumes additional time, referred as communication time in this paper. The communication time is dependent on data size exchanged and the available wireless bandwidth. We estimate the computation and communication needed for the tasks and choose to execute them on robot processors or servers to minimize the total execution time, in order to satisfy real-time constraints.**

Index Terms— **mobile robots, surveillance and tracking, computation offloading, moving object recognition**

I. Introduction

Video surveillance, moving object recognition, and robotic platforms are the fields that have been extensively studied in recent years. Most of the existing studies on surveillance systems use stationary cameras and can track objects only within limited ranges. Moving objects can be tracked by placing cameras on mobile robots. These robots can be used in a variety of fields such as recognizing and tracking suspicious persons or objects. In this paper, we present a real-time mobile robot surveillance system to recognize and track moving objects. Our system has five modules: (1) image capture, (2) motion detection, (3) object recognition, (4) binocular stereovision, and (5) path-planning and tracking. The block-diagram of this system is shown in Figure 1.

In our system, the robot captures images at regular intervals, checks for a moving object, recognizes and tracks the moving object. The gray colored modules in Figure 1 have to be performed on the robot because the cameras are mounted on the robot and obstacles are detected by laser and sonar sensors of the robot. For the other modules, our method decides where to execute the computation. In this paper, we use the terms *modules* and *tasks* interchangeably. In our method, the tasks of image capture, motion detection, stereovision, and path-planning are executed in a periodic manner after every image is captured; hence these tasks have deadlines. Object recognition also has to be performed before the object moves out of the surveillance range. The deadlines for the tasks are governed by factors such as frequency of image capture and surveillance range. We use the terms *deadlines* and *real-time constraints* interchangeably.

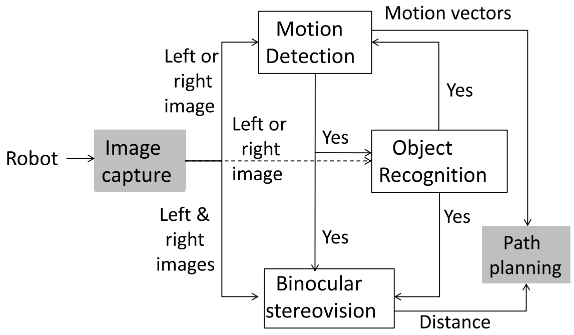


Fig. 1. Block diagram of our system. The gray modules: image capture and path planning, are performed on the robot. Other modules can be performed either on the robot or the server. The dashed line indicates that input from image capture to object recognition does not intersect other lines.

In most existing systems, the computation is performed entirely on robots or entirely on dedicated servers. These approaches have the following limitations: (a) *Entirely on robots:* Mobile robots have low-end processors; therefore the execution time is often too long to meet deadlines. (b) *Entirely on dedicated servers:* The systems in which the entire computation is performed on the servers are affected by the variations in wireless bandwidths. At high bandwidths, data from robots are transmitted to the servers faster; hence communication overhead is less. At low bandwidths, communication time can become the dominant overhead.

In our system, the robot recognizes and follows a moving object. As the robot moves, the amount of computation to recognize objects varies because the backgrounds change and images with different complexities are captured. If the images are more complex, more features are needed to discern the object; hence more computation is needed. Robot processors, with limited computing capabilities, may not always execute tasks within real-time constraints due to variable amounts of computation. With the recent advances in cloud computing [7], servers with scalable computing capability are available. Hence, migrating computation to these servers reduces the computation time. The mechanism of migrating computation to servers to reduce computation

The authors are with the School of Electrical and Computer Engineering, Purdue University, IN, USA {ynimmaga, kumar25, yunglu, csglee}@purdue.edu

This work was supported in part by the National Science Foundation under Grant CNS 0855098. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

978-1-4244-6676-4/10/$25.00 ©2010 IEEE

time is called “computation offloading” [8]. Computation offloading, however, has communication overhead. We estimate the computation and communication involved in each of the tasks. Based on this estimation, we choose to perform a task either on the robot or the server such that the execution time is minimized and the tasks satisfy real-time constraints.

We believe this is the first study to use computation offloading for applications with variable computation and real-time constraints. Our contributions include: (1) We propose a method to adaptively estimate the computation based on the complexity of images captured. (2) We analyze the effect of wireless bandwidths on data exchange between the robot and the server. (3) Based on the computation and communication of the tasks, we present an offloading framework to reduce the execution time, while satisfying real-time constraints.

II. Related Work

Many studies have been conducted on moving object recognition and object tracking. The majority of these studies consider the two modules separately. Each module requires heavy computation, hence consuming tremendous amounts of time on robot processors. We propose a computation offloading framework to divide the computation between robot processors and servers to achieve real-time performance.

A. Object Recognition

Object recognition usually uses the following steps: (1) feature extraction, (2) training, and (3) testing. Features extracted from Haar wavelets [12], intensity/color histograms [14], and visual cortex models [11] are widely used for recognizing objects. Serre et al. [11] use Gabor filters to model visual cortex for recognizing objects. They show that the visual cortex model outperforms many methods in recognizing multi-class objects. We use the same method to recognize objects in our system. The feature extraction and training are performed offline in our method. The feature database is stored in both robots and servers.

B. Moving Object Tracking

Several methods have been proposed for tracking moving objects. Schulz et al. [10] track multiple objects with a mobile robot. Chen et al. [1] present real-time tracking of a single moving object. Gohring et al. [4] use multiple robots for finding positions of moving objects. Kobilarov et al. [5] present an algorithm to track people outdoors. Qian et al. [9] propose a probabilistic approach for simultaneous robot localization and tracking people. Chen et al. [2] use a background subtraction method for real-time tracking. These systems track and follow moving objects or recognize single-class objects. In contrast, we present a real-time tracking system with multi-class object recognition.

C. Computation Offloading

Migrating computation was proposed in [8]. Computation offloading was used in grid computing to perform collaborative (and thus faster) computation. Offloading is emerging as a solution to bridge the gap between the limited computational capabilities of mobile systems and demand for increasingly complex functionalities. Existing studies use computation offloading to reduce energy consumption [6] or to reduce execution time [13]. We believe this is the first study to use computation offloading for applications with real-time constraints in mobile robots.

Technique	Processing Location	Real time	Object Recognition	BA	CA
[10]	server	Yes	None	No	No
[1]	server	Yes	None	No	No
[4]	robot	No	None	No	No
[5]	robot	No	single-class	No	No
[9]	server	Yes	single-class	No	No
[2]	server	Yes	None	No	No
Our method	server or robot	Yes	multi-class	Yes	Yes

TABLE I

Comparison of our system with existing studies (BA: bandwidth adaptive, CA: computation adaptive).

D. Our Contributions

Table I shows the comparison of our system with existing systems. Existing systems use single-class object recognition. They transmit data from robots and use servers for computation. They assume that the wireless bandwidths are always sufficient for exchanging data in real-time. However, the wireless bandwidths may vary due to signal strength attenuation or channel contention. These systems also do not consider tasks with variable amounts of computation. Our system differs from existing studies with the following contributions: (1) We present a real-time tracking system with multi-class object recognition. (2) We consider tasks with variable amounts of computation; we estimate the computation and communication of the tasks before executing them. (3) We present an offloading decision framework for different amounts of computation and wireless bandwidths.

III. Offloading Decisions for Moving Object Recognition and Tracking

Our system consists of five modules as shown in Figure 1. Image capture and path-planning are performed on the robot. The other modules: motion detection, stereovision, and object recognition may execute on the robot or on a server. The data exchange between these modules is small. For example, the data sent from motion detection to object recognition is only one bit: the bit 1 (shown as *yes* in the figure) is sent if motion is detected and the object recognition starts, otherwise 0 is sent. Hence, we consider these modules independently.

The decision of where to execute the modules depends on their amounts of computation and communication. We first estimate the computation involved in the modules, then estimate the communication consumed by offloading these tasks to servers. Based on this estimation, we decide where to execute the tasks such that they satisfy real-time constraints. Motion detection and stereovision have fixed amounts of computation for a given image resolution. We determine the relationship between the amounts of computation for these two modules and show that their offloading decisions are related. Under some conditions, elaborated in Section III-C, the same offloading decision holds for both motion detection and stereovision. Object recognition, however, has a variable amount of computation; hence we estimate the computation and communication to make a decision, irrespective of the other two modules' decisions. We first describe the setup of our system in III-A and identify the real-time constraints. Next, we provide an offloading decision framework for motion detection, stereovision, and object recognition.

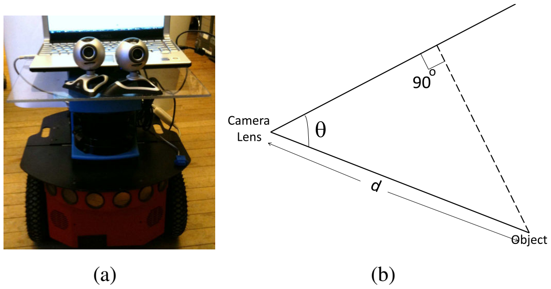
A. System Setup

Our system consists of a pioneer 3DX mobile robot and an on-board computer with a Intel core2-duo 2 GHz processor and two stationary cameras, each with a resolution of 640 × 480. We use a Intel Xeon Linux server, with eight quad-core 2.33 GHz processors and 8 GB RAM. When parallel tasks are executed on the server, we achieve a speedup of about 20 times. The robot setup is shown in Figure 2 (a). The robot is stationed at the surveillance location and images are captured at a frequency of 20 images/sec to detect motion. Once motion is detected, the features of the object are extracted and compared against the categories in a target database. If the object belongs to the target database, the location of the object is computed using stereovision. The robot then follows the object by continuously capturing images and computing the location of the object. To detect motion and to recognize the object, images from one camera are used, whereas for stereovision, images from both cameras are used. We use the following assumptions in our system: (1) The velocity of the moving object is less than the maximum velocity of the robot. (2) There is only a single moving object.

The tasks involved in moving object recognition and tracking are governed by deadlines. Motion detection and stereovision have to finish execution by the time the next image is captured. The deadline for motion detection is t_s, the image capture period (1/20th sec in our system). The selection of t_s depends on the algorithm used to detect motion, and is described in Section III-B. Stereovision is performed after motion detection to compute the object's distance. Hence, the deadline for stereovision is $t_s - t_{md}$, where t_{md} is the motion detection time. Object recognition has to finish execution before the object moves out of camera's view. If the camera's angle of view is θ, and the object is first identified at a distance d, the minimum distance it can travel before it moves out of camera's view is $d \sin\theta$ as shown in Figure 2 (b). If the velocity of the object is v_t, the deadline for object recognition is $\frac{d \sin\theta}{v_t}$.

B. Motion Detection

We use background subtraction to detect motion [15] by comparing adjacent frames and identifying differences. We use rectangles to identify the regions with differences because computing exact contours of objects is computation-intensive. Since the robot follows the moving object, the background also changes continuously. Hence, several blocks with differences are observed. In order to detect the motion of the object while ignoring the background change, we capture images at a high frequency (20 images per second). By using high frequency to capture images, the changes in the background become insignificant and the motion of the object is detected. The robot moves with the object; hence small motion of the object results in smaller changes in background and the object can still be tracked. The movement of the center of the rectangle with the largest area gives the direction of object's movement.



(a) (b)

Fig. 2. (a) Robot setup, (b) Camera's angle of view.

Symbols	Meaning	Units
α	background complexity	-
γ	database size	-
η	speed-up factor of the server	-
β	wireless bandwidth	kbps
θ	camera angle of view	degrees
v_t	velocity of the target object	m/sec
t_s	image capture sampling time	sec
f_r	robot processor speed	GHz
N	number of pixels	-
d_f	file size of features	kB
d_I	file size of image	kB
t_{md}, t_{sv}, t_{or}	execution times of motion detection, stereovision, object recognition	sec
k_{md}, k_{sv}, k_{or}	computation/pixel of motion detection, stereovision, object recognition	cycles

TABLE II
TABLE OF SYMBOLS

The computation involved in motion detection c_{md} depends on the resolution of the images. As the resolution increases, more pixels are compared to identify regions with differences. The computation is given by $c_{md} = k_{md} \times N$, where k_{md} is an algorithm-specific constant that represents the computation per pixel (in cycles) for motion detection and N is the total number of pixels. We first estimate the computation at the instruction level. Our algorithm performs the following instructions per pixel on average: two load instructions to read the pixels from the current and the previous frames, one subtraction to determine the difference between these pixels, one store to write the difference, one load to scan the difference image, one store to write motion blocks. The number of cycles per instruction (CPI) is a platform-dependent constant; therefore we compute k_{md} as the product the number of instructions and the CPI of the target platform. When the entire computation is performed on a robot processor with speed f_r, the time consumed by motion detection $t_{md,r}$ is given by:

$$t_{md,r} = \frac{c_{md}}{f_r} = \frac{k_{md} \times N}{f_r}. \quad (1)$$

If the entire computation is offloaded to a server, with an effective speed-up factor of η, the computation time becomes $\frac{t_{md,r}}{\eta}$ seconds because the images can be divided into several parts and the differences can be computed in parallel. However, transmitting images to the server consumes an additional $\frac{d_I}{\beta}$ seconds for every image of file size d_I at a wireless bandwidth of β. The total time $t_{md,s}$ required for motion detection, when offloaded to the server is given by:

$$t_{md,s} = \frac{c_{md}}{\eta \times f_r} + \frac{d_I}{\beta}. \quad (2)$$

We offload the computation to the server when $t_{md,r} > t_{md,s}$ as shown in Equation (3). If this inequality does not hold, we perform motion detection on the robot. Hence the time for motion detection is the smaller value between $t_{md,r}$ and $t_{md,s}$. If the smaller value exceeds the deadline t_s, we skip motion detection for the next sample to avoid accumulation of delay for later samples.

$$\frac{k_{md} \times N}{f_r} > \frac{k_{md} \times N}{\eta \times f_r} + \frac{d_I}{\beta}. \quad (3)$$

C. Binocular Stereovision

We use binocular stereovision [3] to compute the distance of an object. The left and right images are rotated and shifted to find matching parts. The shifted amount in pixels is called *disparity*. We use the disparity map to compute the distance of the object. The actual distance x of an object is computed as $x = \frac{y \times i}{j}$, where y is a known distance of an object, i is the disparity map intensity of the object, and j is the disparity map intensity of the object whose distance is measured [3].

The amount of computation required by stereovision c_{sv} is proportional to the resolution of the images. As the resolution increases, more pixels are used to compute disparity maps. The computation is given by $c_{sv} = k_{sv} \times N$, where k_{sv} is an algorithm-specific constant that represents the computation per pixel (in cycles) and N is the total number of pixels. At the instruction level, our algorithm needs 8 loads, 5 stores and 4 arithmetic operations per pixel on average, to read both left and right images, rotate and shift them, and compute the disparity. The value of k_{sv} is computed similar to k_{md} as the product of the number of instructions and the CPI. For our algorithms, we observe that $k_{sv} \approx 3 \times k_{md}$.

The analysis for offloading stereovision is similar to the motion detection. Similar to Equation (3), stereovision is offloaded to the server if $t_{sv,r} > t_{sv,s}$, where $t_{sv,r}$ and $t_{sv,s}$ are the execution times on the robot and the server respectively. The computation for stereovision is larger than motion detection because $k_{sv} > k_{md}$. The communication for stereovision is twice that of motion detection ($\frac{2d_I}{\beta}$) because two images are sent to the server. Similar to Equation (3), stereovision should be offloaded if:

$$\frac{k_{sv} \times N}{f_r} > \frac{k_{sv} \times N}{\eta \times f_r} + 2 \times \frac{d_I}{\beta}. \quad (4)$$

By comparing Equations (3) and (4), we infer that stereovision is always offloaded, when motion detection is offloaded, provided stereovision consumes at least twice the amount of computation as motion detection ($k_{sv} \geq 2k_{md}$). This eliminates an additional step of decision for stereovision. However, if the motion detection is not offloaded, nothing can be inferred about the decision for stereovision; hence Equation (4) is used for the decision. If stereovision consumes less than twice the amount of computation as motion detection ($k_{sv} < 2k_{md}$), we perform stereovision on the robot, if motion is detected on the robot. However, if the motion detection is offloaded, we use Equation (4) for the decision. Table III summarizes the relationship between offloading decisions of motion detection and stereovision. In our system, $k_{sv} > 2k_{md}$; hence we always offload stereovision, when motion detection is offloaded.

Condition	Motion Detection	Stereovision
$k_{sv} \geq 2k_{md}$	offload robot	offload Equation (4)
$k_{sv} < 2k_{md}$	offload robot	Equation (4) robot

TABLE III

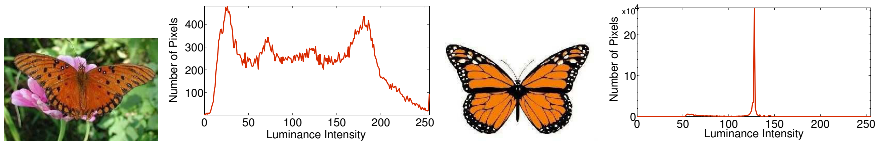
OFFLOADING RELATIONSHIP BETWEEN MOTION DETECTION AND STEREOVISION.

The time for stereovision is the smaller value between $t_{sv,r}$ and $t_{sv,s}$. If the smaller value exceeds the deadline $t_s - t_{md}$, we skip stereovision for the next sample to avoid accumulation of delay for later samples.

D. Object Recognition

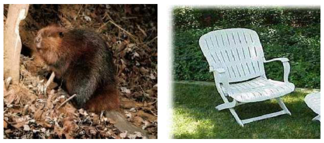
We recognize objects using visual cortex model [11] in two steps: (1) *feature extraction* and (2) *search* through the feature database. Features are extracted by applying a set of Gabor filters with different scales and orientations to images and combining their outputs to form complex vectors [11]. Serre et al. [11] show that 16 Gabor filters model the visual cortex sufficiently. The computation of feature extraction is proportional to the number of features n_f. As the robot moves, n_f varies because images with different complexities are captured; these images require different numbers of features for discerning objects. The amount of computation for search depends on the number of features compared and the size of the object database γ. The computation is given by $c_{or} = c_f + c_s = k_f \times n_f + k_s \times n_f \times \gamma$, where c_f and c_s are computations of feature extraction and search, k_f and k_s are algorithm-specific constants for computation per feature (in cycles) for feature extraction and search, computed similarly to k_{md} and k_{sv}.

Existing studies treat images with different complexities equally; hence the same computation is consumed. However, as the robot moves, object recognition requires variable amounts of computation. If the same n_f is used for all the images, performance of object recognition deteriorates for small n_f's and real-time constraints may not be met for large n_f's. Hence, we develop a metric called "background complexity" (α) to estimate the complexity of images and



(a) (b) (c) (d)

Fig. 3. (a) Image with background clutter, (b) Luminance histogram of (a). The histogram is dispersed, (c) Image without background clutter, (d) Luminance histogram of (c). The histogram has long narrow peaks. Images are taken from Caltech 101 object database.



(a) (b)

Fig. 4. Object and background with (a) similar colors, (b) different colors. Images belong to Caltech 101 database.

determine n_f adaptively for different images. The complexity α arises from two factors namely clutter (α_c) and similarity (α_s) and is given by $\alpha = \frac{\alpha_c + \alpha_s}{2}$.

Clutter α_c: The difficulty of discerning an object from its background increases with the background clutter. Figures 3 (a) and (c) show images with and without background clutter. We quantify the amounts of clutter as the dispersion of the images' luminance histograms, measured using a statistical metric called quartile coefficient of dispersion measured as follows: The data are sorted in increasing order of their magnitudes and three numbers from the sorted data are selected such that they divide the data into four almost equal sums. These numbers are called quartiles. Quartile coefficient is defined as $\frac{q_3 - q_1}{q_1 + q_3}$, where q_1, q_2, and q_3 are the three quartiles. Quartile coefficient of dispersion ranges between 0 and 1. Figures 3 (b) and (d) show luminance histograms of (a) and (c). Figure 3 (b) has short dispersed peaks (α_c = 0.8) whereas (d) has long narrow peaks (α_c = 0.1).

Similarity α_s: The complexity also increases with the similarity between the colors of object and background. We consider all the pixels in the rectangle determined by the motion detection as the object. We use correlation between chrominance histograms of the object and its background to quantify the similarity. The value of α_s ranges between 0 (no correlation) and 1 (100% correlation). Figures 4 (a) and (b) show two images with and without background similarity (α_s = 0.79 and 0.02 respectively).

The value of α ranges between 0 and 1, because $\alpha = \frac{\alpha_c + \alpha_s}{2}$. For images with different values of α, we need different numbers of features to achieve a given accuracy of classification [11]. The classification accuracy is defined as the ratio of number of objects similar to the query object to the total number of objects in the category. We use 20 categories from Caltech101 object database, each category containing 200 to 500 images. To obtain a classification accuracy of at least 90%, Figure 5 shows the number of features required for different α's. Using regression, we find the following relationship between n_f and α. The value of n_f ranges between 1135 (α = 0) and 4881 (α = 1).

$$n_f = -16089\alpha^4 + 30411\alpha^3 - 14268\alpha^2 + 3692\alpha + 1135 \quad (5)$$

The time taken by the robot processor $t_{or,r}$ to perform object recognition is given by:

$$t_{or,r} = \frac{c_{or}}{f_r} = \frac{c_f + c_s}{f_r} \quad (6)$$

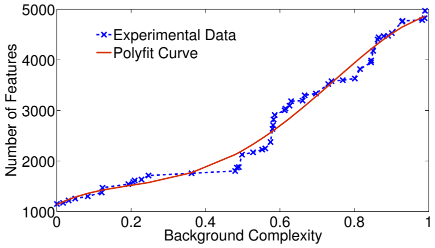


Fig. 5. Number of features for different background complexities to achieve 90% accuracy.

If the entire object recognition is offloaded to the server, the computation time becomes $\frac{t_{or,r}}{\eta}$ secs. However, transmitting images to the server consumes an additional $\frac{d_I}{\beta}$ secs. The total time $t_{or,s}$, when offloaded to the server is given by:

$$t_{or,s} = \frac{c_f + c_s}{\eta f_r} + \frac{d_I}{\beta} \quad (7)$$

We also consider a scenario in which the object recognition is partially offloaded to the server. In this scenario, feature extraction is performed on the robot and search is offloaded to the server. The total time $t_{or,p}$ for partial offloading is given by:

$$t_{or,p} = \frac{c_f}{f_r} + \frac{c_s}{\eta f_r} + \frac{d_f}{\beta} \quad (8)$$

where d_f is the file size of features. We take the smaller value among $t_{or,r}$, $t_{or,s}$, and $t_{or,s}$. If $t_{or,r}$ is smaller, we perform the entire computation on the robot. If $t_{or,s}$ is smaller, we offload the entire computation to the server. If $t_{or,p}$ is smaller, we offload the computation partially to the server. If the smaller value is greater than the deadline $\frac{d \sin\theta}{v_t}$, we turn the robot in the direction of object and obtain more time to recognize the object.

IV. Experiments and Results

Our system has many parameters such as the distance of the object, the speed of the object, the camera's angle of view, the image capture frequency, the wireless bandwidth, the server speed-up, and the amount of computation. We first show a base case with fixed parameters, and analyze how offloading decision is affected by varying the wireless bandwidth, the server speed-up, the background complexity and the object database size, by varying one at a time.

A. Base Case

Consider a robot stationed at the surveillance location. A moving object is spotted at a distance of 2m from the camera

t_s	d	v_t	$t_{md,r}$	$t_{sv,r}$	$t_{or,r}$	α	η
0.05s	2m	1.5m/s	0.06s	0.2s	1.7s	0.5	10

β	γ	d_I	d_f	$t_{md,s}$	$t_{sv,s}$	$t_{or,p}$	$t_{or,s}$
50kbps	1000	20 kB	10 kB	0.41s	0.82s	0.55s	0.57s

TABLE IV

VALUES FOR BASE CASE.

with an angle of view 90°. The velocity of the object is 1.5m/sec and the maximum velocity of the robot is 1.6m/sec. The execution times for motion detection and stereovision on the robot are 60 msec and 200 msec respectively. The object recognition time on the robot with a background complexity 0.5 is 1.7sec (0.2sec for feature extraction and 1.5sec for search) for a database containing 1000 images. The execution times on the server for these modules are computed using Equations (3), (4), (7) and (8). The values for the base case are listed in Table IV. Motion detection and stereovision are performed on the robot and object recognition is partially offloaded for the base case.

B. Wireless Bandwidth

Wireless bandwidth (β) available to the robot may vary due to signal strength attenuation and channel contention. At low bandwidths, data exchange between the robot and the server becomes a dominant overhead. This communication is negligible at high bandwidths. Figures 6 (a), (b), and (c) show the execution times and offloading decisions for motion detection, stereovision, and object recognition for the base case. The maximum wireless bandwidth observed in our lab is 140 kbps. We vary the wireless bandwidth from 10 kbps to 140 kbps by introducing delay in data exchange. The amount of computation does not vary with wireless bandwidths. Motion detection consumes small amounts of computation; hence communication becomes the dominant overhead at low bandwidths. Motion detection is performed on the robot for the available range of wireless bandwidths. Stereovision is performed on the robot for bandwidths up to 110 kbps and on the server for bandwidths greater than 110 kbps. Object recognition consumes heavy computation; hence for the given database size, we observe that partial offloading is the best choice for wireless bandwidths up to 60 kbps and full offloading above 60 kbps. The execution times of motion detection and stereovision exceed deadlines and are avoided for later samples until they finish execution. Bandwidths as high as 54 Mbps can be achieved using 802.11g; at such bandwidths, communication time becomes nearly zero and the decisions will be dependent on computation alone.

C. Server Speed

As the server speed (η) increases, computation time reduces. We use a server with eight quad-core 2.33 GHz processors and an on-board computer with one 2 GHz dual core processor. Theoretically, maximum speed-up that can be achieved for parallel applications, considering only the processors, is $\frac{2.33\times8\times4}{2\times2} = 18.64$. Experimentally, we find that we achieve about 20 times speed up because of factors such as cache, pipeline, and memory. We use the base-case and vary the server speeds by turning off some processors. Figures 6 (d), (e), and (f) show the execution times and offloading decisions of motion detection, stereovision, and object recognition for different speed-ups. We observe that motion detection and stereovision are performed on the robot at all the speeds because offloading consumes a tremendous amount of communication time at a bandwidth of 50 kbps. Object recognition benefits from partial offloading at this bandwidth for the given server speeds. Almost infinite speed-up can be achieved by using cloud computing; in such cases, the decision will depend on communication alone.

α	γ	β	Offload Decision
0	100	$\beta \leq 100$ kbps	robot
	1000	$\beta \leq 6$ kbps	robot
		$\beta > 6$ kbps and $\beta \leq 100$ kbps	partial
	5000	$\beta \leq 100$ kbps	partial
0.5	100	$\beta \leq 65$ kbps	robot
		$\beta > 65$ kbps and $\beta \leq 100$ kbps	server
	1000	$\beta \leq 6$ kbps	robot
		$\beta > 6$ kbps and $\beta \leq 65$ kbps	partial
		$\beta > 65$ kbps and $\beta \leq 100$ kbps	server
	5000	$\beta \leq 65$ kbps	partial
		$\beta > 65$ kbps and $\beta \leq 100$ kbps	server
1	100	$\beta \leq 35$ kbps	robot
		$\beta > 35$ kbps and $\beta \leq 100$ kbps	server
	1000	$\beta \leq 35$ kbps	partial
		$\beta > 35$ kbps and $\beta \leq 100$ kbps	server
	5000	$\beta \leq 35$ kbps	partial
		$\beta > 35$ kbps and $\beta \leq 100$ kbps	server

TABLE V

OFFLOADING DECISIONS FOR OBJECT RECOGNITION FOR DIFFERENT BANDWIDTHS, BACKGROUND COMPLEXITIES, AND DATABASE SIZES.

D. Background Complexity

Object recognition has a variable amount of computation. As the background complexity (α) increases, more features are required for object recognition. The execution times on both the robot and the server increase with the background complexity. We use the base-case and vary α by considering different images. Figure 7 (a) shows the execution times and offloading decision for object recognition. For images with $\alpha \leq 0.55$, object recognition benefits from partial offloading. For other images, the entire computation is offloaded to the server because of heavy computation resulting from larger numbers of features. The computation is significant even when $\alpha = 0$, because 1135 features (Equation 5) are still used for recognizing objects.

E. Database Size

The computation of object recognition also depends on the size of the object database (γ). We use the Caltech 101 database for our experiments. We vary γ between 20 and 5000 images. As γ increases, search consumes higher computation time. Figure 7 (b) shows the execution times and offloading decision for object recognition for different database sizes. For $\gamma \leq 150$, the base-case is performed on the robot processor. For $\gamma > 150$, object recognition benefits from partial offloading. The computation time for feature extraction remains the same, because it is not dependent on the database size. As the database size increases, the feature extraction time becomes negligible; hence the decision depends

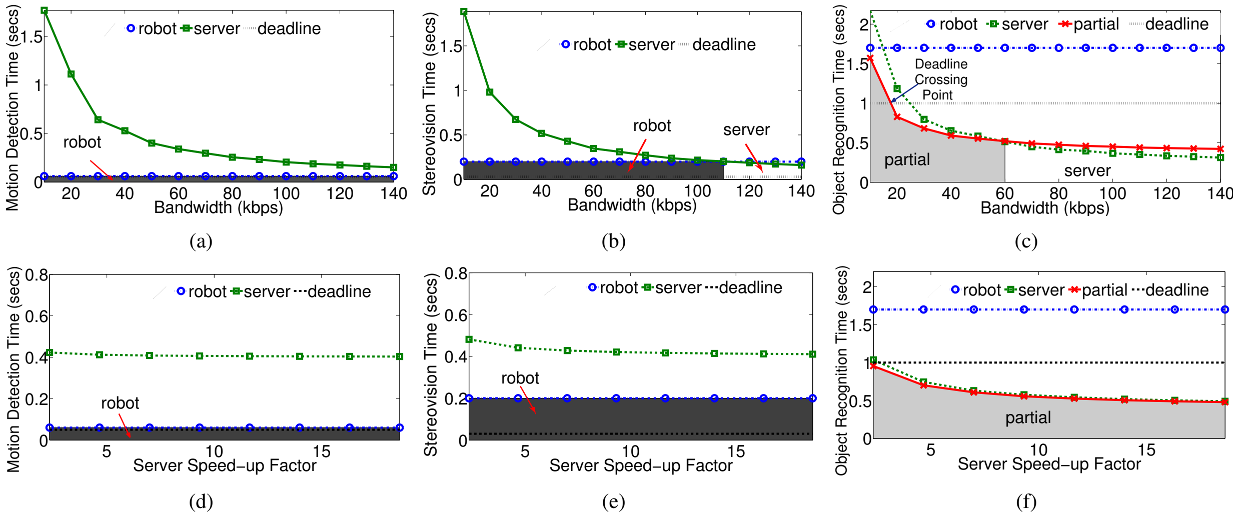


Fig. 6. (a),(b),(c) Execution times and offloading decisions for motion detection, stereovision, and object recognition for different wireless bandwidths. (d),(e),(f) Execution times and offloading decisions for motion detection, stereovision, and object recognition for different server speeds. The dark gray regions represent computation on the robot, light gray regions represent partial offloading, and white regions represent full offloading to the server.

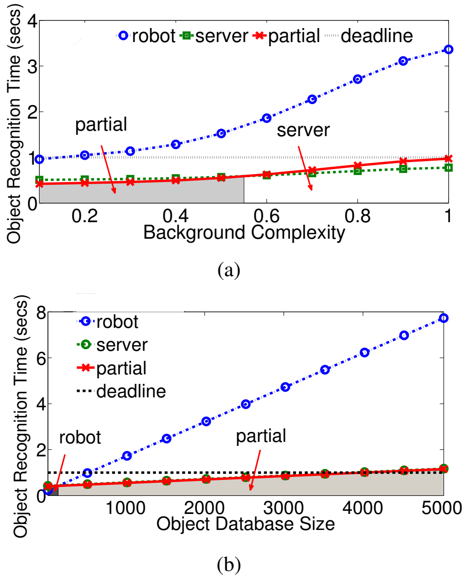


Fig. 7. Offloading analysis for object recognition for different (a) background complexities, (b) database sizes. The dark gray regions represent computation on the robot, light gray regions represent partial offloading, and white regions represent full offloading to the server.

on the communication required for partial offloading and full offloading. In the base-case, partial offloading consumes less communication time. Therefore, object recognition benefits from partial offloading.

We observe the offloading decisions of object recognition by varying background complexity, wireless bandwidth and database size together. Table V shows the offloading decision for object recognition for different α's, β's, and γ's. For low background complexities and database sizes, the computation is low; object recognition is performed on the robot. For high background complexities and database sizes, computation is high; object recognition is offloaded to server. For intermediate values, object recognition benefits from partial offloading.

V. Conclusion

We present a real-time moving object recognition and tracking system. We estimate the computation and communication involved in the tasks. We develop an offloading decision framework that divides the computation between the robot and the server. We present an analysis of the effects of wireless bandwidth, server speed-up, image complexity, and object database size on offloading decisions.

References

[1] Chen et al. A Moving Object Tracked by A Mobile Robot with Real-Time Obstacles Avoidance Capacity. In *International Conference on Pattern Recognition*, 2006.

[2] Chen et al. BEST: A Real-time Tracking Method for Scout Robot. In *IEEE/RSJ IROS*, 2009.

[3] Cullen et al. Stereo Vision Based Mapping and Navigation for Mobile Robots. In *IEEE ICRA*, 1997.

[4] Göhring et al. Multi Robot Object Tracking and Self Localization Using Visual Percept Relations. In *International Conference on Intelligent Robots and Systems*, 2006.

[5] Kobilarov et al. People Tracking and Following with Mobile Robot Using an Omnidirectional Camera and Laser. In *International Conference on Robotics and Automation*, 2006.

[6] Li et al. Computation Offloading to Save Energy on Handheld Devices: a Partition Scheme. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2001.

[7] Nurmi et al. The Eucalyptus Open-Source Cloud-Computing System. In *International Symposium on Cluster Computing and the Grid*, pages 124–131. IEEE Computer Society, 2009.

[8] Powell et al. Process Migration in DEMOS/MP. *ACM SIGOPS Operating Systems Review*, 17(5):110–119, 1983.

[9] Qian et al. Simultaneous Robot Localization and Person Tracking using Rao-Blackwellised Particle Filters with Multi-modal Sensors. In *IEEE/RSJ IROS*, 2008.

[10] Schulz et al. Tracking Multiple Moving Objects with a Mobile Robot. In *Conference on Computer Vision and Pattern Recognition*, 2001.

[11] Serre et al. Object Recognition with Features Inspired by Visual Cortex. In *IEEE CVPR*, pages 994–1000, 2005.

[12] Wang et al. Wavelet-based Indoor Object Recognition through Human Interaction. In *International Conference on Advanced Robotics*, 2003.

[13] Xian et al. Adaptive Computation Offloading for Energy Conservation on Battery-Powered Systems. In *ICPDS*, 2007.

[14] David Lowe. Object Recognition from Local Scale-Invariant Features. In *International Conference on Computer Vision*, 1999.

[15] M. Piccardi. Background Subtraction Techniques: A Review. In *International Conference on Systems, Man and Cybernetics*, 2004.