Real-Time Face Detection and Tracking Utilising OpenMP and ROS

Eduardo Tusa
Ingeniería Civil
Universidad Técnica de Machala
Machala, Ecuador
etusa@utmachala.edu.ec

Arash Akbarinia
CVC
Universitat Autònoma
Barcelona, Spain
arash.akbarinia@cvc.uab.es

Raquel Gil Rodriguez
IP4EC
Universitat Pompeu Fabra
Barcelona, Spain
raquel.gil@upf.edu

Corina Barbalata
Ocean Systems Laboratory
Heriot-Watt University
Edinburgh, UK
cb237@hw.ac.uk

***Abstract*—The first requisite of a robot to succeed in social interactions is accurate human localisation, i.e. subject detection and tracking. Later, it is estimated whether an interaction partner seeks attention, for example by interpreting the position and orientation of the body. In computer vision, these cues usually are obtained in colour images, whose qualities are degraded in ill illuminated social scenes. In these scenarios depth sensors offer a richer representation. Therefore, it is important to combine colour and depth information. The second aspect that plays a fundamental role in the acceptance of social robots is their real-time-ability. Processing colour and depth images is computationally demanding. To overcome this we propose a parallelisation strategy of face detection and tracking based on two different architectures: message passing and shared memory. Our results demonstrate high accuracy in low computational time, processing nine times more number of frames in a parallel implementation. This provides a real-time social robot interaction.**

***Keywords*-RGB-D; Kinect; Human Detection and Tracking; ROS; OpenMP.**

I. Introduction

As robots become involved into the daily life, the social interactions between humans and robots are becoming essential. Due to that robots need to understand and react intelligently to the actions and intentions of multiple humans in a visual scene [1]. The first step is human localisation, for example by applying the algorithms proposed in [2], [3].

Moreover, estimation of head pose and gaze direction are required to determine the visual attention of an interaction partner [4]. Humans use these non-verbal cues to express communicative acts and to get visual focus of attention [5]. Interpretation of these behaviours is a challenging task for machines. The head pose estimation techniques can be classified into feature- and appearance-based methods. The former methods consider low-dimensional features, e.g. the position of facial elements, in different image regions, which can be computed geometrically [6], and trained by an artificial neural network (ANN), or any other non-linear regression methods. The later methods consider the entire region of face. One way to recover the head pose is to apply coarse-to-fine classifiers, which are previously trained on the pose space [7].

Computer vision algorithms suffer under ill illuminated scenes [8]. In these environments depth information is an important cue that has several advantages over 2D intensity images. For instance, range images offer a simple representation of 3D information and are robust to the changes in colour and illumination. To benefit from this 3D information, social robots can be equipped with the depth sensors, such as Kinect, thanks to their easy accessibility.

Face detection and tracking are computationally demanding tasks. For example, Kinect acquires 30 frames per second, that only allows 33 milliseconds to process each frame. Therefore, parallel computation - use of two or more processors in combination to solve a single problem [9] - in on-line task execution is essential for real-time-ability of a social robot [10]. The implementation of parallel approach for face detection has been treated widely in the literature. For instance, Shekar et al. [11] propose data partition by using shared memory multicore systems. This method exploits the fact of multiple cores communications through shared caches. Fifteen sub-sampled images per frame are split across multiple cores to search for possible face candidates.

This paper presents a novel human detection and tracking method based on the depth and RGB images of Kinect. The main idea is to estimate the social scene of a bar/pub/café environment crowded with multiple subjects. To accomplish this goal, the Robot Operating System (ROS) [12] is used as a platform of communication. Furthermore, we propose efficient parallelisation strategies based on two different architectures: (i) message passing and (ii) shared memory architecture. OpenMP API is used, which allows incremental and scalable parallelisation of existing code, providing thus a fast way of parallelising the face detection system with minimal changes to its data structures and algorithm [13].

The paper is organised as follows: Section II and III describes the system, and explains the rationale of the parallelisation architectures respectively. Section IV discusses the experimental results. Finally, Section V concludes the paper and provides possible directions for future work.

II. System overview

Our system is divided into four important stages: human detection in (1) depth and (2) RGB images, (3) face tracking, and (4) estimation of head position and torso orientation (Figure1). The inputs to our system are depth and RGB data

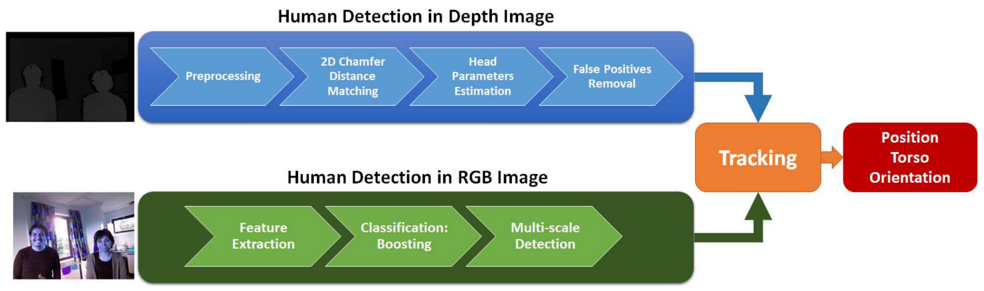

Human Detection in Depth Image
Preprocessing
2D Chamfer Distance Matching
Head Parameters Estimation
False Positives Removal
Tracking
Position Torso Orientation
Human Detection in RGB Image
Feature Extraction
Classification: Boosting
Multi-scale Detection

Figure 1: Block diagram of the implemented algorithm.

of the same image. The outputs are the (x, y, z) coordinates of the face position, and the yaw angle, θ, that represents the torso orientation. The detection generates a group of possible head candidates that are used to initialise and perform the tracking. During the tracking operation, the target from each scene is tracked along the sequence of the video, generating an estimation of head position that is used to compute the torso orientation.

A. Human detection in depth images

Four phases are carried out in order to detect a human: (1) preprocessing, (2) 2D chamfer distance matching, (3) head parameters estimation and (4) false positives removal.

1) Preprocessing: Kinect provides depth images whose pixel values have been distorted due to the presence of noise introduced by illumination conditions (i.e. shadows) or object occlusions. To overcome these limitations inpaiting is applied in order to fill the image gaps and to propagate the gray levels and sharp details [14].

2) 2D chamfer distance matching: In this phase, we extract the edges of the depth image to compute the distance image, where every pixel gives the distance to the nearest edge. Next, we apply template matching, using a contour representation of an average head, to find the possible regions that may indicate the existence of a person [8]. By constructing an image pyramid we achieve scale invariance. Each image is sub-sampled to generate the next image at the higher level in order to match the template with all scales.

3) Head Parameters Estimation: Each generated region from the previous stage can be approximated by a circle. We follow the described relation between the height of the head and the depth associated to a detected region [8],

$$y = p_1 x + p_2 x^2 + p_3 x + p_4, \tag{1}$$

where $p_1 = -1.38 \times 10^{-9}$, $p_2 = 1.84 \times 10^{-5}$, $p_3 = 0.091$ and $p_4 = 189.38$. The radius of the head, R, is estimated based on the standard height, h, generated by $R = 1.33h/2$. Based on this information, the head is expected to be encountered within a circular region defined by radius R in the edge image. All the candidates that meet the mentioned criterion and pass a certain size threshold are nominated as potential human heads.

4) False positives removal: We considered three criterion to eliminate false positives:

- Approximating contours by polygons: represented by circulars or semi-circulars polygons.
- Merging overlapped heads: the Euclidean distance among close potential heads is computed.
- Matching with the 3D template: the depth image and a 3D template of a human bust is used.

B. Human detection in RGB images

This stage is based on the work of Viola and Jones [2]. In this object detection framework, a window of the target size is slid over the input image, and the Haar-like features are calculated. Because these features are only weak learners a large number of them is necessary to describe an object with sufficient accuracy. Therefore, the Haar-like features are organised in something called "cascade" and fed to an AdaBoost classifier.

C. Tracking

We employed the particle filter [15] which is one of the most used tracking algorithms described in 2.

Algorithm 1 Particle filter tracking

```
1) Select the person to be tracked.
2) Obtain the colour-depth model of the target person.
3) Initialise weighted sample set.
4) Compute colour-depth distribution at sample locations.
5) Compare distributions using Bhattacharyya coefficient.
6) Use Bhattacharyya distance to assign weight to sample.
7) Position estimation using weighted samples.
8) Samples selected based on re-sampling.
```

D. Torso orientation

The yaw angle, θ, that describes the torso orientation is generated by using the information of the depth image I_{depth}. The input of this stage is the centroid coordinates (x_c^I, y_c^I) of detected faces. We computed the θ as follows:

III. Parallelisation implementation

We exploited two different architectures to parallelise our application: (**i**) message passing and (**ii**) shared memory architecture.

Algorithm 2 Computation of the torso orientation θ

```
1) Find the top boundary of the person $(x_c^I, y_c^I - d)$ by subtracting adjacent pixels values and thresholding this difference.
2) Define a point in the torso region $(x_c^I, y_c^I + 1.5d)$.
3) Find the side boundaries of the human torso by subtracting adjacent pixels values and thresholding the difference.
4) Define a rectangle that encloses the detected person based on the generated boundaries.
5) Compute two significant points $p_1^I = (x_1^I, y_1^I)$ and $p_2^I = (x_2^I, y_2^I)$, based on the width of the rectangle.
6) Transform these significant points in the real world, $p_1^W = (x_1^W, y_1^W, z_1^W)$ and $p_2^W = (x_2^W, y_2^W, z_2^W)$.
7) Compute the angle of torso orientation $\theta$, 2.
```

$$\theta = atan\left(\frac{x_2^W - x_1^W}{I_{depth}(p_2^I) - I_{depth}(p_1^I)}\right) \tag{2}$$

A. Message passing

ROS provides a suitable platform for a message passing architecture through a graph of nodes [12]. Nodes communicate with each other via messages[1]. This communication happens in *roscore*, which is a global or local network. The high-level software architecture of the proposed application can be observed in Figure2. We designed three primary nodes:

- *social_robot_depth* detects the faces in the depth image. This topic subscribes to the depth images of Kinect and publishes the detected faces as a list of *Region of Interest* (ROI).
- *social_robot_rgb* detects the faces in the colour image. This topic subscribes to the colour images of Kinect and publishes the detected faces as a list of *ROI*.
- *social_robot_track* which is responsible of tracking the detected faces. This topic subscribes to the "detection" nodes and publishes the tracked faces as a list of *ROI*.

B. Shared memory

Within each node of ROS, OpenMP is used to parallelise the process with a shared memory model.

1) Depth detection: Template matching in every scale is computationally expensive, as it must be calculated for every pixel of the image. In order to speed up this process, the *matchTemplate* function of OpenCV and the false positive removal part of this stage were parallelised. Furthermore, detection at each scale can be processed in its own thread and the results are merged.

2) Tracking: The particle filter was used in order to track all the detected persons. The algorithm can be parallelised by computing the weights of the particles in different threads. The weights were written in a shared memory. Additionally, since the application is multiple people tracking, each tracking is performed in its own thread that brings the benefit of linear speed up. Moreover, since there are no data dependencies among the particles during the computation of the likelihoods, after the histogram distributions are computed, each likelihood can be computed in parallel as long as the processing elements have access to the common reference histogram [16].

[1]Each node can publish a topic and subscribe to the topics of other nodes.

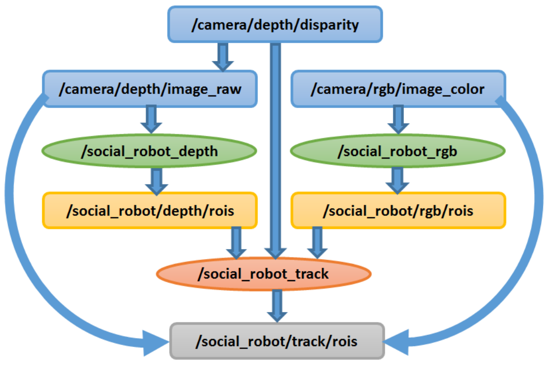


Figure 2: ROS nodes and their communications. We have designed three primary nodes, *social_robot_depth*, *social_robot_rgb*, and *social_robot_track*, which are responsible of detecting faces in depth image, detecting faces in colour image and tracking the detected faces respectively.

C. Advantages of parallel implementation

1) Performance: Image processing applications are computationally very demanding. In the proposed application two different types of images are available: depth and colour images and each frame has to be processed in real-time. The proposed design allows to process each image in its own independent process and different components cooperate at runtime. Since in the detection phase they do not share any data a message passing architecture is chosen.This modularity increases the speed of the application significantly, i.e. if detections and tracking are performed in the same process only 10% of the frames can be processed; however with the proposed design over 80% of the frames can be processed in real-time. In addition to the speeding factor, the designed architecture gives the opportunity to add new sensors and algorithms. The process time increases slightly due to network communication, however this increase can be neglected in comparison with the computational time required by an algorithm.

2) Availability: Since the core components are executed in independent processes, if one component crashes or takes a great amount of time to process one frame, other components do not have to wait. This means each component performs its task and is not hindered by other components.

3) Modifiability: One of the greatest challenges of parallel processes in a shared memory architecture is modification of one process. The scope of change may becomes too large and therefore the modification becomes too cumbersome.

However, in a message passing architecture, this issue does not exist. As long as the type of passed messages are known, any changes within each process stays local. This is one of the main reasons to choose a message passing model for the overall architecture instead of shared memory model.

4) Portability and distributability: Shared memory architecture tends to change its behaviour on different machines and therefore are not very portable. In contrast to that, message passing architecture is not significantly influenced by the host machine and can be easily ported. Additionally, shared memory model can be hardly distributed among machines, whereas message passing architecture are designed to be distributed.

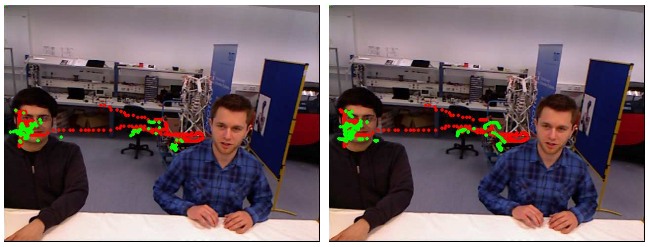
IV. Results and discussions

We tested our proposed algorithm on a laptop *Lenovo IdeaPad Z565* with a processor *AMD Turion II P520 Dual-Core / 2.4 GHz.*

A. Message passing

The application was evaluated with three different videos which had 3508, 1785, and 1774 frames respectively. The application streams the frames as live video and this makes the computation time to be a crucially important parameter. If the algorithm takes long time to process a frame, the detection and tracking will not be able to be applied to all the frames and important information might be lost, thus reducing the accuracy of the system. Since this is an online application, the number of frames per second remains constant, however depending on the speed of the algorithms the number of frames that are processed can vary.

For the evaluation we used two different set-ups. First, all the major components (detection in depth and colour images and tracking) were executed as a standalone application, i.e. in one sequential process. Second, each component was executed as its own process and ROS topics were used to communicate among nodes. The results of these experiments are illustrated in Figure 3. The number of processed frames are summarised in Table I and Figure 4. As it can be observed from Table I and Figure 4 if the



(a) Sequential, processed 420 frames (b) Parallel, processed 3102 frames

Figure 3: Results of detection and tracking. Green our result, red ground truth.

Table I: Number of frames processed for a standalone application versus a distributed system.

		1 Process	3 Processes
	# frames	# processed frames	
Scenario 1	3508	420	3102
Scenario 2	1785	115	1463
Scenario 3	1779	115	1422

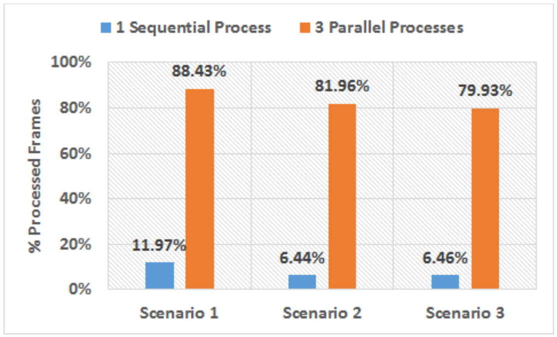


Figure 4: Percentage of processed frames, one versus three processes.

detection and tracking are run in a classical sequential code on average 9.2% of the frames can be processed. However, 83.3% of the frames can be processed when detection and tracking are executed in their own independent processes and communicate with each other via message passing. This is due to the high computational demand of each component. On average, to perform face detection in a depth image with the proposed algorithm takes about 0.125 seconds while the face detection in colour image takes about 0.076 seconds. In addition to that, tracking one person with 200 particles takes about 0.013 seconds. The results show that nine times more information can be processed when detection and tracking are running in parallel and communicate via message passing.

B. Shared memory

In this subsection, the result of our shared memory parallelisation are presented.

1) Template matching: The parallel approach of the *matchTemplate* of OpenCV does not significantly speeds-up the process. The main reason for this is the nature of the sequential implementation of this algorithm. The image is broken down into different blocks, and the results are merged into a final matrix. The merging must be performed in a *critical* region. Entering and leaving a critical section takes time and since this operation is performed too often for merging the program performance is decreased.

2) Chamfer matching: A parallel implementation of the chamfer matching algorithm performs template matching in different scales. Every scale is being executed in one thread. The final results are stored in a *vector*. The sequential version of chamfer matching takes about 0.062 seconds.

After parallelisation, 0.038 seconds are needed to perform the matching, which is about 55% of the sequential version. This is the maximum speed-up we can reach with our set-up because: (**i**) the first scale takes about the same time as all other scales together, and (**ii**) the final results must be merged into one *vector*. Therefore, we have to introduce a *critical* region, which slows down the process. Furthermore, the time to perform the template matching can differ depending on the number of people in an image. The more potential heads in an image, the more time consuming is this stage. As an improvement, the maximum number of features in every scale, e.g. 100 can be set. This approach can statically allocate memory for the final *vector* before compilation. One of the greatest time consuming elements is when size of a *vector* changes.

3) False positive removal: In order to remove the false positives, the contours and their approximate polygons have to be computed. This part can be parallelised to speed up the process. The average time consumed in every frame for this part in case of sequential is about 0.028 seconds. After parallelisation, the time was reduced to an average of 0.015. This is about 50% faster. The reasons why linear reduction of time is not obtained are as follow:

- The results of this part is stored in a dynamic vector, therefore the size changes during runtime. This slows up the process as we have to introduce *critical* section, when each thread is writing the results into that vector, within the parallel region.
- OpenMP does not allow to *break out* of a for loop. This means that a "work around" was introduced by creating a few new flags.
- Last part of false positive removal consist of merging rectangles that are located close by. This part is complex to be parallelised due to nature of the algorithm.

4) Tracking: On average for a video of three persons, the entire tracking component, including data association, calculating new position, and deleting those with low confidence, takes about 0.046 seconds. The core of tracking itself is parallelised, i.e. each person is tracked and its new position is calculated in its own thread. For the same data, we could reduce the time of tracking component to 0.034 seconds. There are number of reasons why parallelisation only reduced the time about 80%. (**i**) Data association and deleting the tracks with low confidence involves in changing the size of *vector*. *C++ vector*s are thread safe as long as their size does not change. Therefore, since in this two parts of tracking the size of the *vector* is changing, it cannot be parallelised. (**ii**) Parallelisation shows its great advantages when there are large number of data. In our dataset, there are only three persons in the video, that does not represent a large data set. Therefore the gains of parallelisation are limited. (**iii**) The weights of our particles are calculated based on the cross-correlation of histograms. Computational

Table II: Time comparison in milliseconds [ms] between Sequential versus parallel implementation of particle filter.

# Particles	Sequential [ms]	Parallel [ms]
200	0.013	0.010
1000	0.142	0.043

time of creating a histogram of a ROI is proportional to size of that region. Therefore, for faces closer to the camera creating histograms is more time consuming, as their size is larger. Whereas, for faces in the back of the scene creating the histograms are less time consuming. Therefore, some threads finish the job faster than the others. And since, tracking component has to wait for all the tracks, the time reduction is not linear, i.e. it is equal to the longest time of all threads. The calculation of each particle is parallelised. TableII illustrates the time difference between the sequential implementation versus the parallel one. As it can be observed from this table the time calculated for particle filter can be speed up significantly when there is a large number of particles. Although the difference for 200 particles is not huge, 1.3% faster, for 1000 particles 3.3 times speed up is observed. This gives the opportunity to increase number of particles in the application and therefore obtain a more accurate tracking.

C. Analysis

1) ROS: ROS provides a good platform for a message passing architecture. However, ROS does not try to specify a threading model for your application. This means that while ROS may use threads behind the scenes to do network management, scheduling, etc., it will never expose its threads to your application. For threading model of our application we utilised OpenMP. As future work, we suggest to parallelise *subscription* and *publications* services and other callbacks. This can be achieving by *MultiThreadedSpinner* of ROS. This requires a careful studying of our algorithm and designing a proper callback queue.

2) OpenMP: Figure 5 illustrates the improvement obtained by utilising OpenMP. As it can be observed a speed up of depth detection and tracking around 1.6 times is obtained. This can naturally be increased if a machine with larger number of core is used. Additionally, parts of the algorithm can be redesigned specifically for parallel implementation. OpenMP does not present good results with object orientation. Since OpenMP is not part of the base language (C/C++), it is difficult for OpenMP to handle class objects. The reason is that the object may not be instantiated at the time the OpenMP "add-on" sees the object. This causes some difficulties such as not being possible to declare object variables as *private* or *shared* within a for loop clause. There is a work around to create a copy constructor for that specific class. However, this can significantly slow down the process, because for every *private* object in a

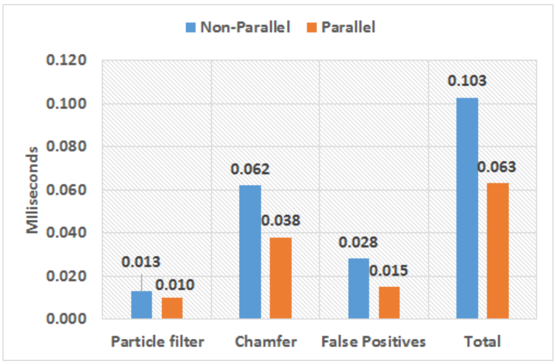


Figure 5: Time improvement obtained by OpenMP.

for loop, compiler has to create a copy of that class and this is a time consuming task. Depending on the tasks performed inside the constructor, compiler has to perform all of those. This includes all the utility and helper classes. In the experiments presented, it was noticed that it is best to declare *default(none)* and handle each variable individually. In general, OpenMP seems to be a powerful library for parallelisation. It works at a fairly coarse-grained level by its ability to perform data decomposition on loops, assigning tasks to individual threads. However, one important aspect that must be taken into account is that in order to use the best of OpenMP one must design the algorithms specifically for parallel implementation. This is a wrong assumption to think first implement sequential and then makes it parallel.

V. Conclusion

In this paper we presented a system for a social robot and evaluated two different strategies of parallelisation, namely *message passing* and *shared memory*. Each method's pros and cons were discussed and we illustrated how to combine these two architecture to significantly speed up a sequential application, without losing the modularity of the implemented code. A message passing architecture is suggested for coarse grain and hared memory for the loop level. The results demonstrate that we can process nine times more number of frames in a parallel implementation. This provides a real-time face detection and tracking.

As future works, we plan to present a comparison study with other similar systems. In addition, to test our proposed approach under different scenarios, including poor lightning conditions to demonstrate the benefits of combining depth and colour information to detect and track humans. Furthermore, to conduct a study to determine the relation between number of cores and computational time.

References

[1] R. Petrick, M. E. Foster, and A. Isard, "Social state recognition and knowledge-level planning for human-robot interaction in a bartender domain," *In Proceedings of the AAAI 2012 Workshop on Grounding Language for Physical Systems.*

[2] P. A. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features," in *CVPR (1)*, 2001, pp. 511–518.

[3] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *International Conference on Computer Vision & Pattern Recognition*, vol. 2, 2005, pp. 886–893.

[4] S. R. Langton, R. J. Watt, and V. Bruce, "Do the eyes have it? cues to the direction of social attention," *Trends in Cognitive Sciences*, vol. 4, no. 2, pp. 50–59, 1980.

[5] A. Gaschler, K. Huth, M. Giuliani, I. Kessler, J. de Ruiter, and A. Knoll, "Modelling state of interaction from head poses for social human-robot interaction," *In Proceedings of the Gaze in Human-Robot Interaction Workshop held at the 7th ACM/IEEE International Conference on Human-Robot Interaction.*

[6] T. Horprasert, Y. Yacoob, and L. Davis, "Computing 3-d head orientation from a monocular image sequence," *In Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference.*

[7] J. R. Meynet, J. Arsan, and J. Thiran, "Fast multiview face tracking with pose estimation."

[8] L. Xia, C.-C. Chen, and J. K. Aggarwal, "Human detection using depth information by kinect," *Workshop on Human Activity Understanding from 3D Data in conjunction with CVPR (HAU3D), Colorado Springs.*

[9] B. Chapman, G. Jost, and R. v. d. Pas, *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation).* The MIT Press, 2007.

[10] J. Bedkowski, "Parallel computing in mobile robotics for rise," *InTechOpen.*

[11] D. Shekhar and K. Varaganti, "Parallelization of face detection engine," *2010 39th International Conference on Parallel Processing Workshops.*

[12] ROS.org, "Documentation," 2012, [accessed 08-December-2012]. [Online]. Available: http://www.ros.org/wiki/

[13] P. E. Hadjidoukas and V. V. Dimakopoulos, "A high performance face detection system using openmp," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 15, pp. 1819–1837, October 2009.

[14] A. Telea, "An image inpainting technique based on the fast marching method," *Journal of Graphics Tools*, vol. 9, no. 1, pp. 25–36, 2003.

[15] M. Goodrum, M. Trotter, A. Aksel, S. Acton, and K. Skadron, "Parallelization of particle filter algorithms," in *Computer Architecture.* Springer, 2012, p. 139149.

[16] H. Medeiros, J. Park, and A. Kak, "A parallel color-based particle filter for object tracking," in *In IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008. CVPR Workshops*, 2008, pp. 1–8.