Distributed Smart Cameras for Hard Real-Time Control

Herwig Guggi
Institute of Networked and Embedded Systems
Pervasive Computing Group
Klagenfurt University
herwig.guggi@uni-klu.ac.at

Bernhard Rinner
Institute of Networked and Embedded Systems
Pervasive Computing Group
Klagenfurt University
bernhard.rinner@uni-klu.ac.at

ABSTRACT

This paper describes an approach for integrating cognition, real-time communication and control. Our test setup is a crane system whose area of operation is observed by distributed smart cameras analyzing the crane's environment. The position and motion information of all detected objects are transferred to the crane controller via a real-time network. With this information the controller can optimize the trajectory of the load. We present preliminary results of a single camera setting and give an outlook for the multi camera data fusion process.

Keywords

smart camera, real-time control

1. INTRODUCTION AND MOTIVATION

Control systems can be found in many real world applications such as industrial automation, automotive and transportation. The controller determines the behavior of the controlled object based on the observed internal state captured by some sensors. In this work we use multiple cameras to observe the environment of the controlled object and provide information of the environments' current state to the controller which is then able to optimize the behavior. Information about the environment, such as position and velocity of obstacles, may be very helpful for control optimization, however to be useful this information has to be delivered within guaranteed time bounds, i.e., in hard real-time. Thus, our ultimate objective is to integrate real-time image analysis, adaptive motion control, and synchronous communication between the imaging and control subsystems.

In this work in progress, we deploy distributed smart cameras [1] to monitor the environment of a model crane. The smart cameras are connected over a time-triggered network with the crane's control system and deliver the position and velocity of all obstacles within the predefined frame rate of 20 fps. By knowing the position of obstacles, the controller can adapt the trajectory of the payload appropriately and increase the safety and efficiency of the crane system.

Our contribution comprises the vision system. This includes the image analysis, the scene reconstruction, the object motion prediction and the communication with the control system.

This paper reports on the current state of this research. The remainder is organized as follows. Section 2 sketches related work. Section 3 briefly describes the system overview and high-level data flow. In section 4 we present preliminary results on a single camera setting and in section 5 we discuss different methods for data fusion in the distributed smart camera network. Section 6 concludes the paper.

2. RELATED WORK

Collision avoidance [2, 3] is a very active field in robotics. Examples include work for ground based robots [4, 5] but recently also for flying robots [6, 7]. However, we only describe closely related work on using multiple, stationary cameras for collision avoidance.

Henrich et al. [8] presented a closely related system where multiple cameras are used to guide a robot in a region with moving obstacles. Obstacle detection is performed with four calibrated cameras connected to a single computer where the position and structure of the obstacles are calculated. After the planning, each configuration (movement step of the robot) is tested for collision with obstacles. If a collision is detected, the trajectory is re-planned. The paper provides no information about the update rate of the scene or if images are captured while the robot is moving.

Work by Ladikos et al. [9] used a similar setup where four cameras are connected to a PC. This PC calculates the 3D model with the help of a GPU (graphics processing unit). Potential collision are checked based on bounding boxes around detected obstacles. To prevent collisions these bounding boxes are extended by a security distance. The cameras operate at a resolution of 1024x768 and at a frame rate of 30 fps.

3. SYSTEM OVERVIEW

Figure 1 depicts the system overview as well as the high-level data flow. The main components include the multi-camera subsystem, the control subsystem (including the crane with its sensors and actuators) and the real-time network for connecting the components. The distributed smart cameras analyse the crane environment. In especially, the cameras perform some object detection individually and fuse this local information within the camera network to determine the

ICDSC 2010 August 31 – September 4, 2010, Atlanta, GA, USA

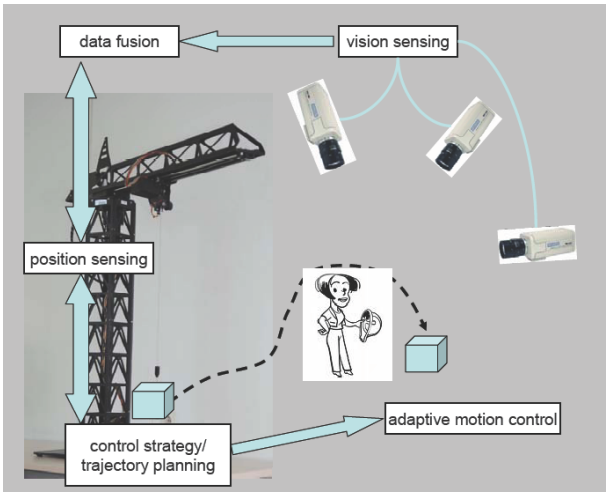

Figure 1: System architecture and high-level data flow

object's position and velocity in the 3D space. This fused information, i.e., object size, position and velocity, is transferred to the control subsystem.

The model crane has a height of approx. 1.3 m and its jib has a length of approx. 1 m. This results in an operation region of the crane in form of a cylinder with radius and height of about 1 m. The ground plane of the observation area is 2 m × 2.25 m which we consider as the environment space. The crane is operated indoors with stable ambient light. The processing time and the position accuracy are the two major requirements for the image analysis on the smart cameras. The control subsystem requires a frame rate of 20 fps which in turn limits the maximum time for processing and network transfer to 50 ms. The objects' positions should be determined within an accuracy of 1 cm.

We use custom-built smart cameras from SLR Engineering which are equipped with an Intel Atom processor running at 1.6 GHz and an 100 MBit Ethernet interface. The camera has a CCD image sensor with a native resolution of 1360 × 1024 pixels. The control subsystem is based on a real-time work station equipped with an Intel Core 2 Duo with 2.13 GHz CPU and 1024 MB RAM. The smart cameras and the control subsystem are connected via TTEthernet [10] which extends classical Ethernet with services based on time-triggered protocols to meet time-critical, deterministic or safety-critical requirements.

3.1 Image Analysis

The image analysis can be basically split into two processes. Functions which have to be performed on the entire image, e.g., image capturing, background subtraction and thresholding are assigned to the first process. These functions are also referred to as "object independent processing"(cp. section 4).

The second part of the image analysis starts with the list of segments that have been detected on the single image. The following steps are dependent on the observed scene and include functions such as transformation to the real-world coordinate system, fusion with corresponding segments from other cameras, object reconstruction, speed determination. All these functions have to executed for each object individually and are also referred to as"object dependent processing".

Table 2: The standard deviation of time for different resolutions

Resolution	σ object independent	σ object dependent
640x480	0.61	0.1
720x576	1.34	0.04
800x600	3.97	0.08
1024x768	3.35	0.06
1360x1020	8.51	0.09

4. SINGLE CAMERA RESULTS

We conducted first experiments with a single camera setting where the camera is mounted at a height of 1.5 meters at a distance of 2 meters from the crane. In this setting we assume that all objects are placed on the ground plane. All processing is performed on the smart camera. For every detected object, the position of the central point at the ground plane and the size of the bounding box are transferred to the control subsystem at every frame.

We evaluated the performance of the single camera setting wrt. the achieved processing speed, position accuracy and communication performance.

4.1 Processing speed

In our image processing pipeline the total processing speed varies with the resolution and the number of detected objects in the field of view. Some functions of the image pipeline are independent of the number of objects while others are not. Object independent execution includes background subtraction, thresholding and contour detection. The function for transferring the scene description is also independent of the number of objects because all position data can be packed into a single Ethernet frame.

Object dependent execution time includes mapping to real-world coordinates, position determination, motion detection, to mention only some. Table 1 lists the measurement execution times for different resolution (first column) and number of detected objects (second column). The following two columns represent the accumulated execution times for the object-independent and the object-dependent functions (median of 120 measurements). The final column presents the computed execution time for 50 objects at the correspondent resolution. This time is calculated with the equation 1 where t_{oi}, t_{od} represent the object-independent and object-dependent time, σ_{oi}, σ_{od} are the object-independent and object-dependent standard deviation values as listed in table 2 and n is the number of objects (second row of table 1).

$$t = t_{oi} + 2 * \sigma_{oi} + \frac{t_{od} + 2 * \sigma_{od}}{n} * 50 \qquad (1)$$

The implementation of the controller is currently able to process the sensory data at a frame rate of 20 fps. Table 1 shows that within 38.67 ms 97.7% ($average + 2 * standard\ deviation$) of the scenes with a resolution of 1024 × 768 can be calculated. If equation 1 is modified to use $3 * \sigma$, 99.9% of the scenes at a resolution of 1024 × 768 can be calculated at 42.2 ms. Thus it is very unlikely that the

Table 1: The processing time compared for different resolutions, measured: median for 120 runs

Resolution	number of objects	object independent processing time	object dependent processing time	time for 50 Objects (calculated)
640x480	12	7.86 ms	2.32 ms	19.59 ms
720x576	13	10.31 ms	2.67 ms	23.54 ms
800x600	15	11.97 ms	3.07 ms	30.67 ms
1024x768	17	19.77 ms	4.03 ms	38.67 ms
1360x1024	19	35.83 ms	4.55 ms	65.3 ms

Table 3: The accuracy listed for different resolutions, accuracy is in [mm/pixel/axis]

Resolution	minimum accuracy	maximum accuracy
640x480	7.22	2.29
720x576	6.45	1.96
800x600	6.14	1.83
1024x768	4.72	1.43
1360x1020	3.59	1.00

processing cannot keep pace with the frame rate.[1]

4.2 Position Accuracy

The position accuracy depends on the geometric setting as well as the resolution of the image. Table 3 lists the minimum and maximum accuracy for each resolution. The accuracy of any pixel is calculated in two steps. In the first step, the real world coordinates of the selected pixel and all of its neighbors is calculated. In the second step, the absolute distance between the real-world-coordinates of each neighboring pixel with the pixel in the center is calculated. The average of these values is the final accuracy as it is listed in table 3.

To decide on a specific setting, the combination of processing speed and position accuracy has to be determined. For example by considering our measurements, we can achieve a total frame rate of 20 fps with a position accuracy of 5 mm per pixel at each axis.

4.3 Protocol performance

The communication protocol is designed to provide the abstracted object information to the control system. The chosen structure consists of a small header (4 Bytes) and a simple representation for the object. The objects are represented either as a cylinder or rectangular box. Each object is defined by its form, its color, the position, speed and orientation all in 3D space. All in all 28 Bytes are sufficient to represent each object. Thus, descriptions of more than 50 objects can be packed into a single Ethernet frame of 1500 bytes.

5. DISTRIBUTED CAMERA PROCESSING

In the work in progress, we use multiple cameras object detection and position determination. On the one hand, multiple cameras with overlapping field of views help to omit some of our initial system requirements, e.g., object placement on the ground plane, dedicated objects' shapes. On the other hand, the complexity of the system increases. All components, i.e., the smart cameras; have to be configured to send the data in the correct format at the correct time to the correct destination.

In a multi camera system, each camera captures the raw frames and performs the preprocessing. This process will include all "object independent" steps (cp. in section 4). A major challenge is the fusion of the results from the individual cameras. One option is to perform the fusion on an additional hardware component. This approach is only necessary if it turns out that the camera platforms do not have enough computing resources to perform the fusion process.

For the current use case we use of 3 smart cameras. There is a number of possible methods for fusing the individual results from each camera. The simplest one is that all cameras perform the preprocessing. After this step, two cameras forward their abstracted data to the third one. This camera will then fuse all data and forward the result to the control system. Currently no time measurement for the fusion process is available, so the extra delay that would result from this structure cannot be assumed yet.

Pipelining is another possible fusion approach. This method starts with each camera capturing the current frame and performing the object-independent processing. After that, the first camera sends its results to the second one. This camera fuses the information from the own frame with the data from the first camera. The result is forwarded to the last camera. This camera is then responsible for the final fusion and to forward the information to the control system. This fusion mechanism has the benefit that the delay between frame capturing and information transmission to the control system is relatively small. A drawback is that the delay between two frame capturing increases with the number of cameras that are included in the system.

Figure 2 shows the idea of an alternative pipelined fusion mechanism. Compared to the previous approach, this method has an increased delay from image capturing to information transmission. The benefit of this system is that the frame rate is increased.

During the project it might turn out that it makes sense to keep the fusion process on a single device. In that case a variation of a loop scheduling can be performed (see figure 3). The first scene is evaluated by the first camera, the second scene by the second camera and the last scene by the last camera. With that structure, each camera can focus the fusion process on a single scene and all scene data can be evaluated from the same device at the same time.

[1]In the rare case the deadline is missed, the objects' position information at the previous sampling time is transferred to the controller which can then decide whether to discard this data.

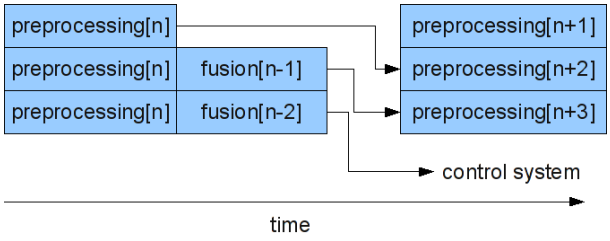

Figure 2: Pipeline fusion approach, increased frame rate

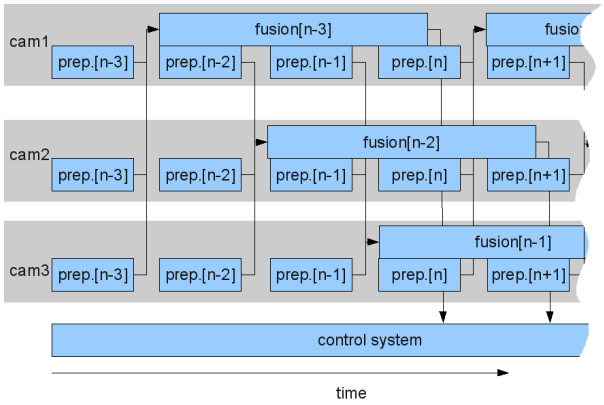

Figure 3: Fusion in a loop

6. CONCLUSION

Smart cameras with integrated image analysis are fast enough to detect objects and convert these information. The information of the detected objects is transferred to the control unit. The control system uses these object information which would otherwise not be available to plan the trajectory of the crane. The detailed and real-time view of the objects in the operating range allows the planning of the trajectory with respect to collision avoidance and optimisation criteria like maximum speed or minimum energy consumption.

Acknowledgment

This work was supported by the Austrian Science Promotion Fund (FFG) under grant 819482.

7. REFERENCES

[1] B. Rinner and W. Wolf, "A Bright Future for Distributed Smart Cameras," *Proceedings of the IEEE*, vol. 96, no. 10, pp. 1562–1564, October 2008.

[2] S. J. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha, and P. Dubey, "ClearPath: highly parallel collision avoidance for multi-agent simulation," in *SCA '09: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. New York, NY, USA: ACM, 2009, pp. 177–187.

[3] S.-E. Yoon, B. Salomon, M. Lin, and D. Manocha, "Fast collision detection between massive models using dynamic simplification," in *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. New York, NY, USA: ACM, 2004, pp. 136–146.

[4] S. Vacek, T. Schamm, J. Schröder, J. M. Zöllner, and R. Dillmann, "Collision avoidance for cognitive automobiles using a 3D PMD camera," in *Intelligent Autonomous Vehicles 2007, Toulouse, France*, 2007.

[5] K.-D. Kuhnert and M. Stommel, "Fusion of Stereo-Camera and PMD-Camera Data for Real-Time Suited Precise 3D Environment Reconstruction," in *IROS*, 2006, pp. 4780–4785. [Online]. Available: http://ieeexplore.ieee.org/iel5/4058334/4058335/04059173.pdf

[6] J. Byrne and C. J. Taylor, "Expansion segmentation for visual collision detection and estimation," in *ICRA'09: Proceedings of the 2009 IEEE international conference on Robotics and Automation*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1938–1945. [Online]. Available: http://jeffreybyrne.com/docs/byrne-icra09.pdf

[7] J.-C. Zufferey, A. Beyeler, and D. Floreano, "Autonomous flight at low altitude with vision-based collision avoidance and GPS-based path following," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2010. [Online]. Available: http://icra2010.grasp.upenn.edu/

[8] D. Henrich and T. Gecks, "Multi-camera collision detection between known and unknown objects," in *2nd ACM/IEEE International Conference on Distributed Smart Cameras ? ICDSC 2008*, vol. 2. Stanford/USA: ACM/IEEE, Sept 7-11 2008, pp. 1–10. [Online]. Available: http://www.ai3.uni-bayreuth.de/resypub/files/ICDSC08.16.mitHeader.kompr.pdf

[9] A. Ladikos, S. Benhimane, and N. Navab, "Real-Time 3D Reconstruction for Collision Avoidance in Interventional Environments," in *MICCAI '08: Proceedings of the 11th International Conference on Medical Image Computing and Computer-Assisted Intervention, Part II*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 526–534.

[10] www.tttech.com.