4

Stereo Vision System for Remotely Operated Robots

Angelos Amanatiadis and Antonios Gasteratos
Democritus University of Thrace
Greece

1. Introduction

Remotely operated robots, functioning in hazardous and time critical environments, have significant requirements for control and visual information (Davids (2002), Murphy (2004)). The control systems are supposed to guarantee a precise timely response in order to prevent fatal scenarios in bomb disposal operations or in life rescue missions. Significant role to these operating scenarios play the concurrent visual information provided to the remote operators by the on-board mounted cameras.
Visual information (Fong & Thorpe (2001), Desouza & Kak (2002)) is often displayed in one or more monitors depending on the number of on-board mounted cameras. In sophisticated and multi-tasking robots more than one operators are performing certain actions. Especially, in case of robots with grippers and robotic arms, one operator might be dedicated only with the maneuvering and controlling of the robotic arms or grippers. In these operation scenarios, the dedicated user must be focused only on this task and furthermore should have the best visual understanding of the working field. The widely used equipment and gear for these assignments is a Head Mounted Display (HMD) and an attached head tracker.
The HMD projects visual feedback of the remote robot in front of operator eyes. A single camera feedback projection in both eyes is not so significant since the result in operator's perception is the same as being watched from a single monitor. Thus, a pair of cameras are used instead, in order to provide a real stereo feedback to the operator's HMD, thus enhancing his visual perception and improving the sense of depth (Willemsen et al. (2004)). Consequently, operators can judge situations and perform actions more efficiently based on the qualitative information of the synchronized stereo video streams. The use of a head tracker expands the operator's functions while it offers a hands-free ability to remotely control the pose of the robot head. The inertial measurement devices used for the head tracking usually contain rate gyroscopes (gyros) and accelerometers. The measurements of the inertial sensor can be processed and transmitted as control signals to the remote robot.
Many different interfaces have been presented in literature recently. A method of robot teleoperation that allows a human operator to control a robot manipulator is presented in (Kofman et al. (2005)). It uses a non-contacting, vision-based, human-robot interface for both the communication of the human motion to the robot and for feedback of the robot motion to the human operator. However, this visual feedback does not give the operator the depth visual information, which is necessary for this critical task. In (Bluethmann et al. (2003)), a sophisticated anthropomorphic robot is developed for space operations. It is comprised of a stereo

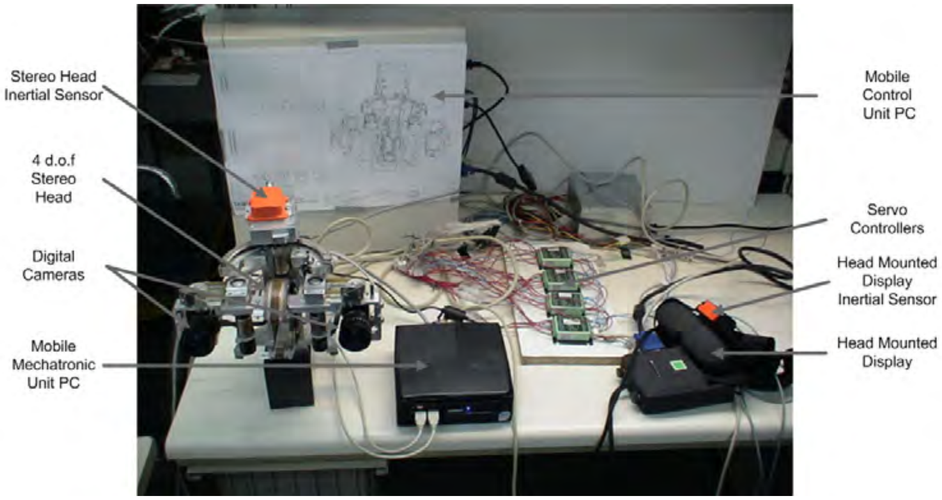


Fig. 1. The Stereo vision system.

head which transmits the video feedback to the operator through a HMD. The same human-machine interface is developed also in (Tachi et al. (2003)) for robot control. Both implementations however, require sophisticated and expensive equipment and are built with proprietary software. In (Marin et al. (2005)), an on-line robot architecture is presented. It enables the control of a robot by interacting with an advanced user interface with very promising results but the real-time constraint for control can not be guaranteed.
In this chapter, we focus on a human-machine interface which guarantees the real-time control of a binocular robotic head. Furthermore, a stereo video streaming transmission with low latency is presented. This hands-free interface is implemented exclusively with open source software on a Linux-based Real-time Operating System. The control and video architecture satisfies also the demands of recent sophisticated telerobotics for flexibility and expandability.

2. System Design

The functions of the stereo vision system, as shown in Fig. 1, are separated into video streaming and motion control and are both implemented with the use of two host computers. The first host computer is called Mobile Mechatronic Unit (MMU), and is placed on the mobile platform where the stereo vision head is operating. The second host computer comprises the Mobile Control Unit (MCU) which is placed on the remote control center. There, the remote operator wears the HMD with the attached head tracker, as shown in Fig. 2.
The video streaming presents high computational burden and resource demand while it requires the full usage of certain instruction sets of a modern microprocessor. In contrast, motion control includes filtering and signal processing from the head tracker output and requires the operation system to be able to execute real-time tasks. This demand for high multimedia performance and real-time motion control is realized by a computer structure consisting of two high performance computers with RT-Linux operating system for the MMU and MCU host computers. However, the two main tasks of the video streaming and the motion control will be performed in different kernel layers due to their different requirements. Furthermore, RT-

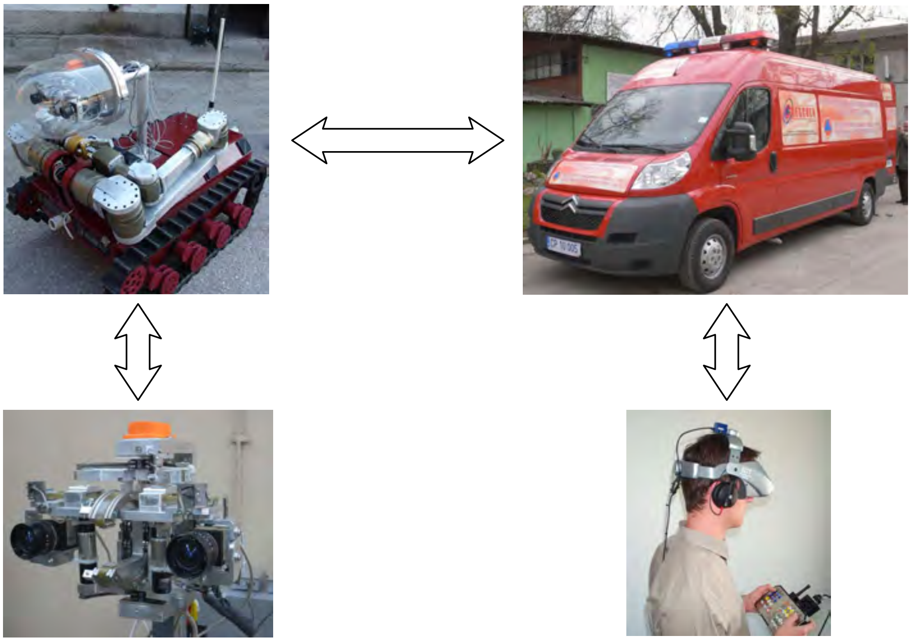


Fig. 2. The remotely operated robot.

Linux operating system was chosen in order to ensure that the different operation and communications loops occur at deterministic rates, and that safety critical tasks are performed reliably. The two host computers are connected together with a wireless high speed network. The communication protocol between the computers uses a higher level abstraction, built on top of sockets, meeting the requirements for low latency and priority handling. Apart from the libraries, the communication protocol consists of a server daemon residing on each side MCU and MMU and acts as gateway to the other side. Each server daemon is in charge of delivering the messages received by the remote end to the clients in its network, and forwarding the messages received by clients in its network, to the remote end, through the wireless link. Furthermore, there is a priority list that refers to the priority assigned to each operation. In the MCU and MMU communication subsystem, the priority handling is realized by means of the cooperation between the server protocol and quality of service features such as packet identification rules, service flow classes for bandwidth reservation, latency and jitter control on the identified packets, and quality of service classes with associated service flow classes. Figure 3 shows the flowchart of the stereo vision software architecture. The right part shows the control flow from the head tracker starting from the MCU and ending to the motors of the stereo head. The left flow shows the video stream from the stereo head cameras of the MMU to the player in the MCU.

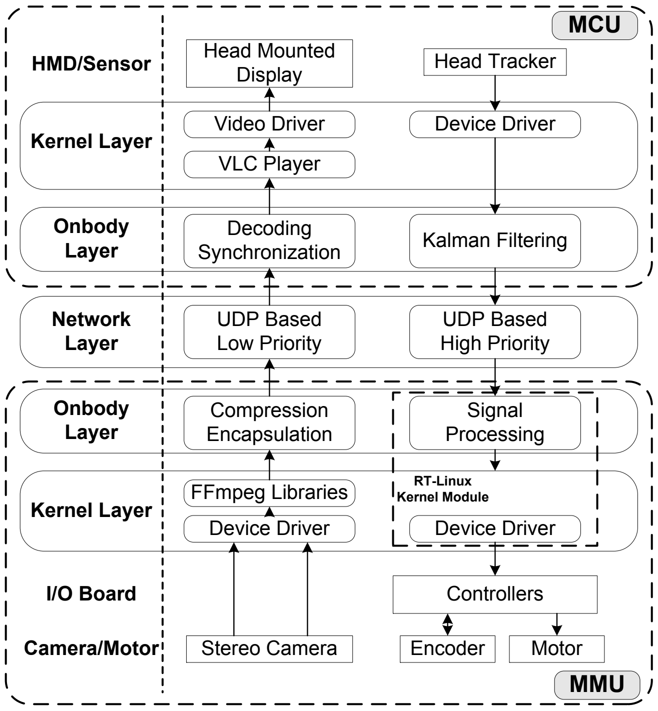


Fig. 3. Flowchart of the stereo vision software architecture.

3. Control System

3.1 Hardware Architecture

A head tracker inertial measurement unit was used to obtain high update rate measurements. Its internal low-power signal processor provides 3D orientation as well as kinematic data of 3D acceleration and 3D rate of turn (rate gyro). The data used for the head tracking is the pitch and yaw in order to send the pan and tilt commands to the teleoperated stereo head, respectively. The chosen interface used for connecting the sensor to the MCU computer is the RS-232, in order to have full access to the basic level of the sensor unit and a full compatibility for the drivers, since no serial-to-USB converter drivers are needed. The second sensor attached on the stereo head mechanism, as shown in Fig. 1, is used for the stabilization of the stereo head (Amanatiadis et al. (2007)). Special attention was paid for the placement of this inertial sensor. Possible errors and distortions from the strong currents of the servo motors can be quite large enough in order to deteriorate the inertial measurements (Roetenberg et al. (2005)). A global reset is performed each time the HMD sensor is initialized to orientate the tracker in such a way that the sensor axes point in exactly the same direction as the axes of the operator′s global coordinate frame. The sample frequency used is 100 Hz with a baudrate of 115 Kbps.

Two harmonic drive actuators are used to move the pan and tilt axis of the stereo head, based on feedback acquired from incremental position encoders. The chosen high precision encoders guarantee a specification 0.01 degree resolution and a maximum frequency response of 100 KHz. The DC servo motors have a maximum output speed of 110 rpm and maximum radial load 59 N, which is adequate for the two cameras load. Each servo is connected to a controller which sends low-level commands to the actuators for executing the trajectories received by the head tracker. A very precise calibration of the controllers was performed so that we could utilize the great degree precision of the position encoders. Position control strategy was chosen while position is the most important aspect of a high performance head tracking control. The Proportional Integral Derivative (PID) controller values were calibrated in discrete-time through the use of real-time processes running with fixed time steps. The use of a simple, and easy to tune control strategy across the pan and tilt axis helped to ensure the reliability and robustness of the whole system. The following equation represents the general PID controller (Astrom & Hagglund (1995)).

$$u = K_p e + K_i \int e dt + K_d \frac{d(-PV)}{dt} \tag{1}$$

Position control requires an additional controller on top of the velocity controller since it sets the desired velocities in all driving phases, especially during the acceleration and deceleration phases. This control procedure has to take into account not only the current speed as a feedback value, but also the current position, since previous speed changes or inaccuracies may have had already an effect on the robot's position. The chosen experimental parameter tuning can be described by the following simple steps (Braunl (2008))

- Selection of the typical operating setting for the desired speed, set to zero integral and derivative parts, and then increase of K_p to maximum or until oscillation occurs.
- Division of K_p by two, when oscillation occurs.
- Slowly increase of K_d while increasing or decreasing the speed. For the smoothest response choose the selected value of K_d.
- Slowly increase K_i until oscillation starts. Then divide K_i by 2 or 3.
- In case the overall controller performance is satisfactorily under the typical system conditions, the tuning is successful.

3.2 Software Architecture

Key feature for the implementation of the real-time control is the used operating system. Since critical applications such as control need low response times, RT-Linux operating system is ideal for both MMU and MCU host computers. The distribution used is an Open Source project which provides an integrated execution environment for embedded real-time applications (Mantegazza et al. (2000)). It is based on components and incorporates the latest techniques for building embedded systems. The architecture is designed to develop hybrid systems with hard and soft real-time activities as shown in Fig. 4. The Linux kernel is treated as the lowest priority task under the RT kernel. In this case, we allocated the critical task of control at the RT-Linux level and the less critical tasks, such as inertial data filtering, at the Linux level. The real-time tasks need to communicate with user-space processes for things like file access, network communication or user interface. Thus, it provides FIFOs and shared memory implementations that provide communication with this user-space processes. The interface for both kinds of activities is a POSIX based interface. Software routines such as boot code, initialization positions, and input/output functions were developed using a combination of hand coded C or assembly language.

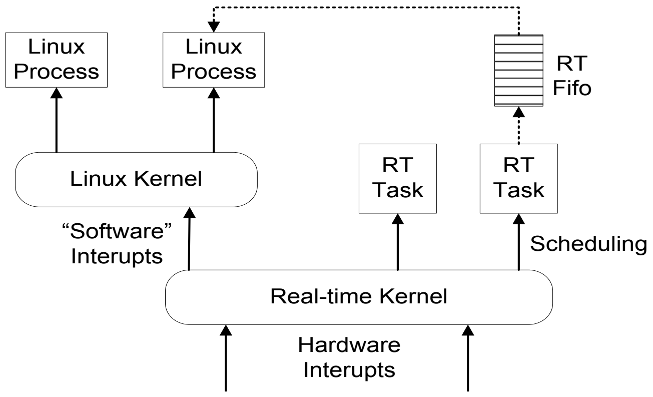


Fig. 4. The chosen operating system combines the use of two kernels, RT-Linux and Linux to provide support for critical tasks and soft real-time applications, respectively.

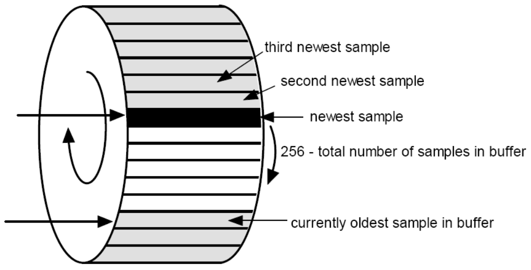

third newest sample
second newest sample
newest sample
256 - total number of samples in buffer
currently oldest sample in buffer

Fig. 5. The polling or event mechanism in the internal buffer of sensor.

The data received by the head tracker at the MCU, was firstly filtered by a Kalman filter. The software strategy dilemma of whether to use polling or events was considered in our implementation. Apart from the fact that the choice is mostly dependent on the user programming environment several other considerations were examined. When using the polling method, the user continuously or at a certain interval, queries the head tracker if new orientation data has been calculated. When queried, the sensor will immediately return the most recently calculated data, as shown in Fig. 5. The polling method is useful when the query function runs in a loop at a certain update rate and each time orientation data is needed, the user just needs the latest data and not necessarily every single sample. When using the events method, instead of continuously querying the sensor, the event notifies the user when new data has been calculated and is available for retrieval with the appropriate functions. For the presented system, the appropriate solution is the polling method since it ensures that the operator always get the latest available orientation data when he asks for it. The polling method allows that the other processes in our software to be asynchronous with the sampling rate of the head tracker itself, and we can synchronize the data with our processes. Furthermore, polling is slightly more straightforward to implement.

The errors in the force measurements introduced by our accelerometer and the errors in the measurement of angular change in the orientation with respect to the inertial space introduced by gyroscopes, were the two fundamental error sources which affected the error behavior of

the operators head trajectory. Furthermore, all inertial measurements are corrupted by additive noise (Ovaska & Valiviita (1998)). The Kalman filter (Welch & Bishop (2001), Trucco & Verri (1998)) was used while is a form of optimal estimator, characterized by recursive evaluation using an estimated internal model of the dynamics of the system. The filtering was implemented on the MCU computer where the inertial sensor is attached, using the soft-real time kernel.
The control data received from the MMU, should be translated into motor commands for the equivalent axis. This operation is time critical since fast and accurate position commands to a remote robot is the only way to guarantee its safe operation. This strategy of considering the head tracker commands time critical and their implementation in the hard real-time kernel, allows the overall system to be flexible in a way that additional future motor commands and even more crucial, like the operation of a gripper, can be implemented easily while satisfying the hard real-time constraints.
The concurrency and parallelism was considered in the programming of the robotic system by using a multi-thread model. The motor run time models are not using the $wait.until.done()$ function, while a change in the operator's field of view indicates that the previous movement should not be completed but a new motion position command should be addressed. The following runtime model was chosen for the motor class:

```
Thread 1 (control)
motor.change_position()
do other things

Thread 2 (monitor)
periodically wake up
read.new_sensor_position()
if (new_sensor_position !=...
...old_sensor_position)
old_sensor_position=motor.change_...
...position.(new_sensor_position)
```

Simultaneous and non-synchronized accesses to the same resources, such as servo motors, was not a set of problems for the implementation while the the pitch and yaw commands would move separately the tilt and pan axis, respectively. However, in case of a future additional operation, such as motor stabilization, the sharing of the same resources would be a great problem. Thus, the software programming infrastructure considered the shared resources and critical sections in order to guarantee the expandability and flexibility of the stereo vision system. The critical sections were easily implemented since the protected operations were limited. However, special attention must be paid since critical sections can disable system interrupts and can impact the responsiveness of the operating system.

4. Video Streaming System

4.1 Hardware Architecture

Each of the stereo head cameras on MMU is capable of outputting progressively images of 640×480 pixel resolution at maximum 30 frames per second. The digital cameras transmit the images over the fast USB 2.0 interface directly to the host's memory without the usage of frame grabbers. In order to determine the internal camera geometric and optical characteristics, camera calibration was necessary. A variety of methods have been reported in the bibliography. The method we used is described in (Bouget (2001)) using its available C Open Source code. The method is a non self-calibrating thus, we used a projected chessboard pattern to estimate the camera intrinsics and plane poses. Finally, the calibration results were used to rectify the images taken from cameras in order to have the best results in possible subsequent image processing algorithms. The video processing requires high computational burden and resources while it makes a full usage of certain instruction sets of a modern microprocessor. Thus, a high performance processor was chosen for the MMU computer. The operator in the MCU receives the stereo pair of images in the stereo HMD. The chosen HMD has the same input resolution like cameras 640×480 and a refresh rate of $70Hz$. Two separate 15 pin D-Sub (VGA) interfaces are used for the stereo image input to the HMD. Thus, the MCU computer is equipped with a double output high performance graphic card in order to display in different outputs each video stream.

4.2 Software Architecture

Vision systems of mobile robots must unify the requirements and demands of both computer vision and image processing disciplines and robotic and embedded system disciplines. While the state of the art in computer vision algorithms is advanced, many computer vision processes are computationally expensive and thus inappropriately for real-time applications. Therefore, the resource demands of computer vision applications are in conflict with the requirements posed by robotics and embedded systems. For this system, a compression scheme must be implemented in order to transmit the stereo image stream. The high input data rate from the cameras of $2(stereo) \times 640 \times 480(resolution) \times 3(color) \times 8(bit\, per\, pixel) \times 25(fps) \cong$ 351 Mbps requires a compression algorithm with high compression ratio, low computational complexity and good output quality. Furthermore, the compressed video should be packetized and streamed over the communication network.

The architecture chosen aims to make the MMU computer a video server which will perform the following primary tasks:

- Capture video from both cameras
- Compress video using a codec
- Packetize the compressed video and attach time stamps within the packets
- Stream the packets over the communication network

For all the previous tasks, apart from capturing, the FFmpeg video open source libraries (FFmpeg project (2008)) were selected. The video server allows multicast transmission while it sends each video stream to a fixed-destination multicast address. The services dealing with each stream, like the video player in the MCU, only have to listen to the appropriate multicast address, so several services can receive the same video stream without increasing bandwidth consumption. The compression was done using MPEG-4 codec, and the transmission of the video streams using the MPEG Transport Stream (Gringeri et al. (1998)). MPEG-TS provides many features found in data link layers, such as packet identification, synchronization, timing (clock references and timestamps), multiplexing and sequencing information. In the architecture chosen, each processing tree is executed within its own thread and is processed in parallel with other source nodes, like the control loop. This framework ensures appropriate synchronization between the image streams. With this framework, the developers do not need to worry about locking issues and synchronization primitives. The UDP communication protocol was used between the two computers while it uses a higher level abstraction, it is built on top of sockets, and meets the requirements for low latency (Traylor et al. (2005)).

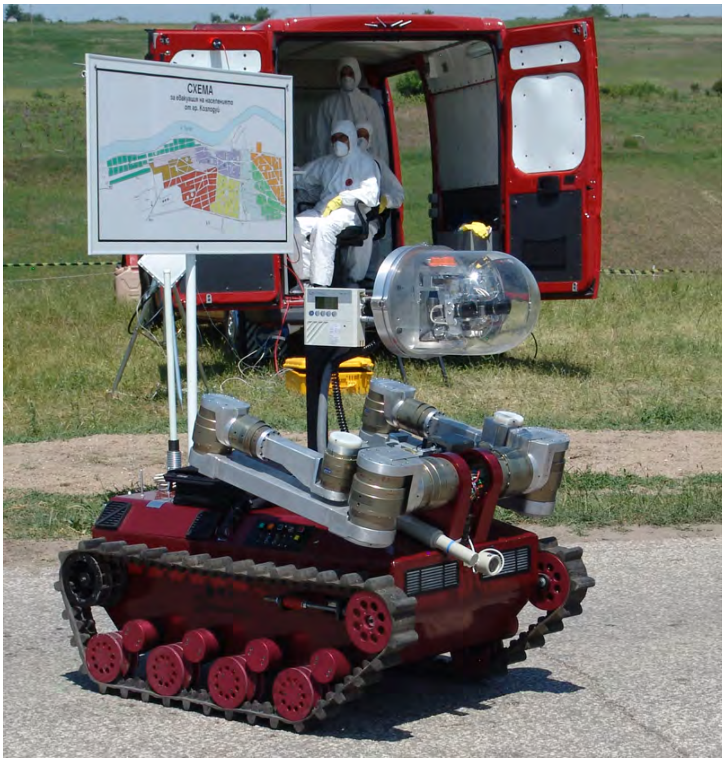


Fig. 6. The remotely operated robot prototype.

For video capturing the video for Linux (Video 4 Linux project (2008)) driver was chosen, which is an open source application programming interface for video capture and output drivers. The available streaming parameters were used to optimize the video capture process as well as the I/O. Our pre-selected video options determine a default number of frames per second in both digital cameras. If less than this number of frames is to be captured or output, applications can request frame skipping or duplicating on the driver side. This is especially useful when using the priority handling of the wireless network topology, where cases of video frames not augmented by timestamps or sequence counters are necessary for bandwidth saving. In order to exchange images between drivers and applications, it was necessary to have standard image data formats which both sides will interpret the same way. The used interface included several such formats but it was not limited only to these formats since driver-specific formats were possible since in the presented stereo vision system, some applications depended on codecs to convert images to one of the standard formats when needed.
The I/O streaming was designed in a way that only pointers to buffers were exchanged between application and driver, ensuring that the data itself was not copied. The capturing application enqueued a number of empty buffers before starting capturing and entering the read loop. The application waited until a filled buffer could be dequeued, and re-enqueued the buffer when the data was no longer needed. Output applications filled and enqueued

the buffers, and when enough buffers were stacked up output was started. In the write loop, when the application run out of free buffers it waited until an empty buffer could be dequeued and reused.

One of the highlights of the presented system is the multiple concepts for real-time image processing. Each image processing tree is executed with its own thread priority and scheduler choice, which is directly mapped to the operating system process scheduler. This was necessary in order to minimize jitter and ensure correct priorization, especially under heavy load situations. Some of the performed image processing tasks were disparity estimation (Georgoulas et al. (2008)), object tracking (Metta et al. (2004)), image stabilization (Amanatiadis et al. (2007)) and image zooming (Amanatiadis & Andreadis (2008)). For all these image processing cases, a careful selection of programming platform should be made. Thus, the open source computer vision library, OpenCV, was chosen for our image processing algorithms (Bradski & Kaehler (2008)). OpenCV was designed for computational efficiency and with a strong focus on real-time applications. It is written in optimized C and can take advantage of multicore processors. The basic components in the library were complete enough to enable the creation of our solutions.

In the MCU computer, an open source video player VLC was chosen for the playback service of the video streams VideoLAN project (2008). VLC is an open source cross-platform media player which supports a large number of multimedia formats and it is based on the FFmpeg libraries. The same FFmpeg libraries are now decoding and synchronize the received UDP packets. Two different instances of the player are functioning in different network ports. Each stream from the video server is transmitted to the same network address, the MCU network address, but in different ports. Thus, each player receives the right stream and with the help of the MCU on board graphic card capabilities, each stream is directed to one of the two available VGA inputs of the HMD.

The above chosen architecture offers a great flexibility and expandability in many different aspects. In the MMU, additional video camera devices can be easily added and be attached to the video server. Image processing algorithms and effects can be implemented using the open source video libraries like filtering, scaling and overlaying. Furthermore, in the MCU, additional video clients can be added easily and controlled separately.

5. System Performance

Laboratory testing and extensive open field tests, as shown in Fig. 6, have been carried out in order to evaluate the overall system performance. During calibration of the PIDs the chosen gains of (1) were $K_p = 58.6$, $K_i = 2000$ and $K_d = 340.2$. The aim of the controlling architecture was to guarantee the fine response and accurate axis movement. Figure 7 shows the response of the position controller in internal units (IU). One degree equals to 640 IU of the encoder. As we can see, the position controller has a very good response and follows the target position. From the position error plot we can determine that the maximum error is 17 IU which equals to 0.026 degrees.

To confirm the validity of the vision system architecture scheme, of selecting RT-Linux kernel operating system for the control commands, interrupt latency was measured on a PC which has an Athlon 1.2GHz processor. In order to assess the effect of the operating system latency, we ran an I/O stress test as a competing background load while running the control commands. With this background running, a thread fetched the CPU clock-count and issued a control command, which caused the interrupt; triggered by the interrupt, an interrupt handler (another thread) got the CPU clock-count again and cleared the interrupt. Iterating the

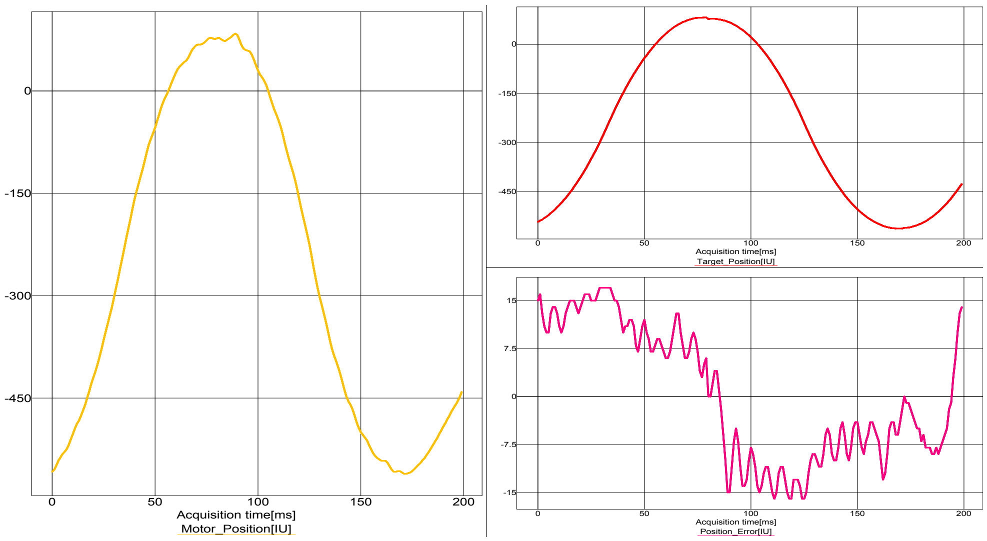
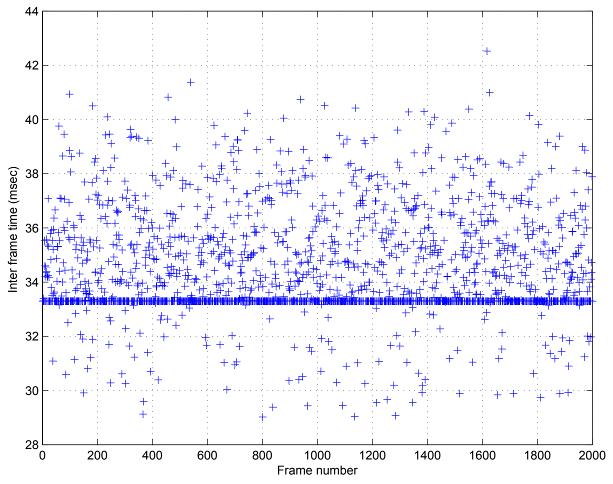


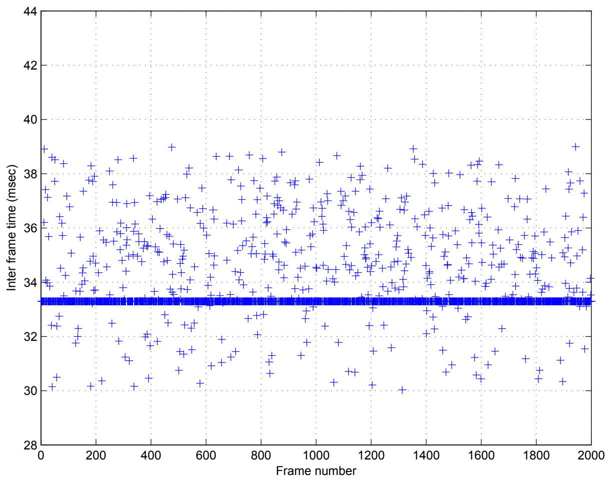
Fig. 7. A plot of position controller performance. Left: The motor position, Up Right: The target motor position, Down Right: The position error.

above steps, the latency, the difference of the two clock-count values, was measured. On standard Linux kernel, the maximum latency was more than 400 msec, with a large variance in the measures. In the stereo vision system implementation in RT-Linux kernel the latency was significantly lower with maximum latency less than 30 msec and very low variation.

The third set of results show the inter-frame times, the difference between the display times of a video frame and the previous frame. The expected inter-frame time is the process period $1/f$ where f is the video frame rate. In our experiments, we used the VLC player for the playback in the MMU host computer. We chose to make the measurements on the MMU and not on the MCU computer in order to calculate only the operating system latency avoiding overheads from communication protocol latencies and priorities. The selected video frame rate was 30 frames per second. Thus, the expected inter-frame time was 33.3 msec. Figure 8(a) shows the inter-frame times obtained using only the standard Linux kernel for both control and video process. The measurements were taken with heavy control commands running in the background. The inter-frame time due to the control process load introduces additional variation in the inter-frame times and increases these times to more than 40ms. In contrast, Figure 8(b) shows the inter-frame times obtained using the RT-Linux kernel with high resolution timers for the control process and the standard Linux kernel for the video process. The measurements were taken with the same heavy control commands running in the background. As we can see, the inter-frame times are clustered more around the correct value of 33.3 msec and their variation is lower.



(a)



(b)

Fig. 8. Inter-frame time measurements: (a) Both control and video process running in standard Linux kernel; (b) Control process running in RT-Linux kernel and video process in standard Linux kernel.

6. Conclusion

This chapter described a robust prototype stereo vision paradigm for real-time applications, based on open source libraries. The system was designed and implemented to serve as a binocular head for remotely operated robots. The two main implemented processes were the remote control of the head via a head tracker and the stereo video streaming to the mobile control unit. The key features of the design of the stereo vision system include:

- A complete implementation with the use of open source libraries based on two RT-Linux operating systems
- A hard real-time implementation for the control commands
- A low latency implementation for the video streaming transmission
- A flexible and easily expandable control and video streaming architecture for future improvements and additions

All the aforementioned features make the presented implementation appropriate for sophisticated remotely operated robots.

7. References

Amanatiadis, A. & Andreadis, I. (2008). An integrated architecture for adaptive image stabilization in zooming operation, *IEEE Transactions on Consumer Electronics* **54**(2): 600–608.

Amanatiadis, A., Andreadis, I., Gasteratos, A. & Kyriakoulis, N. (2007). A rotational and translational image stabilization system for remotely operated robots, *Proc. of the IEEE Int. Workshop on Imaging Systems and Techniques*, pp. 1–5.

Astrom, K. & Hagglund, T. (1995). *PID controllers: Theory, Design and Tuning*, Instrument Society of America, Research Triangle Park.

Bluethmann, W., Ambrose, R., Diftler, M., Askew, S., Huber, E., Goza, M., Rehnmark, F., Lovchik, C. & Magruder, D. (2003). Robonaut: A robot designed to work with humans in space, *Autonomous Robots* **14**(2): 179–197.

Bouget, J. (2001). Camera calibration toolbox for Matlab, *California Institute of Technology, http//www.vision.caltech.edu* .

Bradski, G. & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*, O'Reilly Media, Inc.

Braunl, T. (2008). *Embedded robotics: mobile robot design and applications with embedded systems*, Springer-Verlag New York Inc.

Davids, A. (2002). Urban search and rescue robots: from tragedy to technology, *IEEE Intell. Syst.* **17**(2): 81–83.

Desouza, G. & Kak, A. (2002). Vision for mobile robot navigation: a survey, *IEEE Trans. Pattern Anal. Mach. Intell.* **24**(2): 237–267.

FFmpeg project (2008). *http//ffmpeg.sourceforge.net* .

Fong, T. & Thorpe, C. (2001). Vehicle teleoperation interfaces, *Autonomous Robots* **11**(1): 9–18.

Georgoulas, C., Kotoulas, L., Sirakoulis, G., Andreadis, I. & Gasteratos, A. (2008). Real-time disparity map computation module, *Microprocessors and Microsystems* **32**(3): 159–170.

Gringeri, S., Khasnabish, B., Lewis, A., Shuaib, K., Egorov, R. & Basch, B. (1998). Transmission of MPEG-2 video streams over ATM, *IEEE Multimedia* **5**(1): 58–71.

Kofman, J., Wu, X., Luu, T. & Verma, S. (2005). Teleoperation of a robot manipulator using a vision-based human-robot interface, *IEEE Trans. Ind. Electron.* **52**(5): 1206–1219.

Mantegazza, P., Dozio, E. & Papacharalambous, S. (2000). RTAI: Real time application interface, *Linux Journal* **2000**(72es).

Marin, R., Sanz, P., Nebot, P. & Wirz, R. (2005). A multimodal interface to control a robot arm via the web: a case study on remote programming, *IEEE Trans. Ind. Electron.* **52**(6): 1506–1520.

Metta, G., Gasteratos, A. & Sandini, G. (2004). Learning to track colored objects with log-polar vision, *Mechatronics* **14**(9): 989–1006.

Murphy, R. (2004). Human-robot interaction in rescue robotics, *IEEE Trans. Syst., Man, Cybern., Part C,* **34**(2): 138–153.

Ovaska, S. & Valiviita, S. (1998). Angular acceleration measurement: A review, *IEEE Trans. Instrum. Meas.* **47**(5): 1211–1217.

Roetenberg, D., Luinge, H., Baten, C. & Veltink, P. (2005). Compensation of magnetic disturbances improves inertial and magnetic sensing of human body segment orientation, *IEEE Transactions on neural systems and rehabilitation engineering* **13**(3): 395–405.

Tachi, S., Komoriya, K., Sawada, K., Nishiyama, T., Itoko, T., Kobayashi, M. & Inoue, K. (2003). Telexistence cockpit for humanoid robot control, *Advanced Robotics* **17**(3): 199–217.

Traylor, R., Wilhelm, D., Adelstein, B. & Tan, H. (2005). Design considerations for stand-alone haptic interfaces communicating via UDP protocol, *Proceedings of the 2005 World Haptics Conference*, pp. 563–564.

Trucco, E. & Verri, A. (1998). *Introductory Techniques for 3-D Computer Vision*, Prentice Hall PTR Upper Saddle River, NJ, USA.

Video 4 Linux project (2008). *http://linuxtv.org/* .

VideoLAN project (2008). *http://www.videolan.org/* .

Welch, G. & Bishop, G. (2001). An introduction to the Kalman filter, *ACM SIGGRAPH 2001 Course Notes* .

Willemsen, P., Colton, M., Creem-Regehr, S. & Thompson, W. (2004). The effects of head-mounted display mechanics on distance judgments in virtual environments, *Proc. of the 1st Symposium on Applied perception in graphics and visualization*, pp. 35–38.