Optimization of Robot Telemonitoring System Software using multi-thread method

Midriem Mirdanies

Research Center for Electrical Power and Mechatronics, Indonesian Institute of Sciences (LIPI), Kompleks LIPI, Komp LIPI Bandung, Jl. Sangkuriang, Gd. 20. Lt. 2, Bandung 40135, Indonesia

Email: midr001@lipi.go.id

Abstract

The processor development today is on multi-core and multi-processor which can be used to a speedup of data processing time compared with one processor core only. One of the main ways that can be used to speed up the data processing time is by using multi-thread. Multi-thread method has been implemented on the robot telemonitoring system based on Graphical User Interface (GUI) which has been developed in Research Center for Electrical Power and Mechatronics, Indonesian Institute of Sciences (LIPI). A part of that requires high processing time and dynamic at the telemonitoring systems are the display of real-time thermal cameras and color camera along with tracking algorithm used, it can be seen from the camera display which less smooth especially on the thermal camera, and some of the process is zero which means the buffer is unprocessed which causing lag. Two threads have been added to process each of the cameras separately. C programming language with the OpenCV library and the Integrated Development Environment (IDE) Qt Creator, has been used to implement this method into an application program. Based on experiments, it can be seen that the display of both cameras could run smoothly, the average processing time faster than sequential, and the absence of a zero process time. The average fps generated is more stable and is at the higher level that is 7.5 fps on the thermal camera and 7.5 - 8 fps on the color camera.

Keywords: *multi-thread, telemonitoring, GUI, Qt creator, c language*

1. Introduction

Processor development today is on a multi-processor, and multi-core processors from the two cores and above, as well as the hyper-threading technology. On desktop processor today, it has been available processor with 10 cores with a total of 20 threads with hyper-threading technology that is Intel CoreTM i7-6950X Processor Extreme Edition [1], while for server processor, has been available processor with 24 cores with a total of 48 threads such as Intel Xeon Processor E7-8890 v4 [2] and Intel Xeon Processor E7-8894 v4 [3]. An example of a quad-core processor architecture can be seen in Figure 1.

In Figure 1 it can be seen that each processor can consist of more than one processor core which each core has individual memory, and can connect to one another using shared memory. This multi-core processor can be used to run multiple programs or threads simultaneously to speed up data processing performance. Techniques to optimizing multi-core processors to speed up data processing on a program are parallel programming or multi-thread. In multi-thread, a program is divided into two or more parts and processed on a different thread.

Some publications have been reported using multi-core processors for specific applications. In image processing, cabaret et al. have implemented the Connected Component Labeling (CCL) algorithm for labeling objects [5], and Mirdanies et al. have also conducted research that utilizes multicore processors using shared memory to speed up the multi-object identification process by the SIFT and SURF methods [6]. Huynh et al. have also used multi-core processors in data mining to speed up mining data process on large datasets using a method called Parallel Dynamic Bit Vector Sequential Pattern Mining (pDBV-SPM) [4].

Publications that used multi-threading methods on multi-core processors i.e. rui et al. to hide the deduplication of I/O latency by using multi-thread deduplication (Multi-Dedup) [7]. Sikora et al. have used multi-thread method in the audio signal field to speed up the beam-tracing simulation [8].

Received : 3 May 2017; Revised : 4 Oct 2017; Accepted : 15 Dec 2017

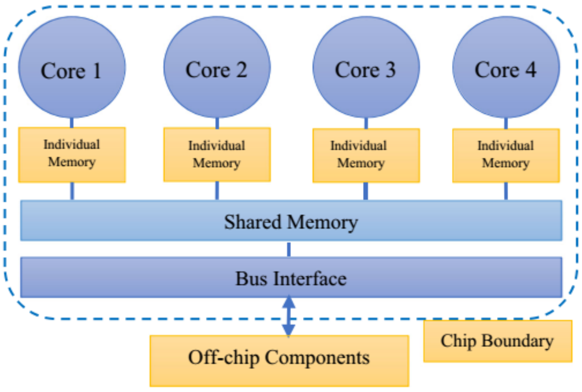


Figure 1. An example of a quad-core processor architecture [4].

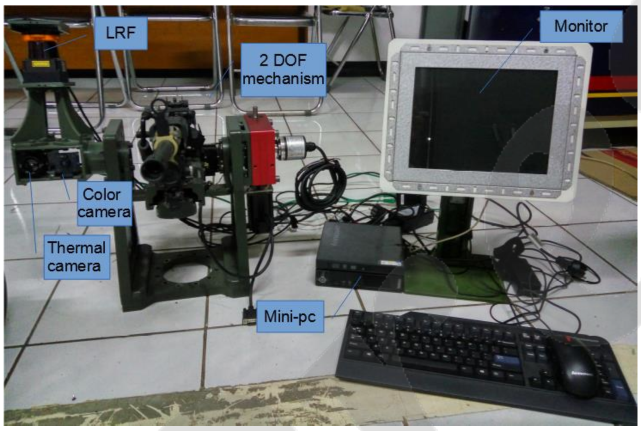


Figure 2. Two DOF of robotic arms.

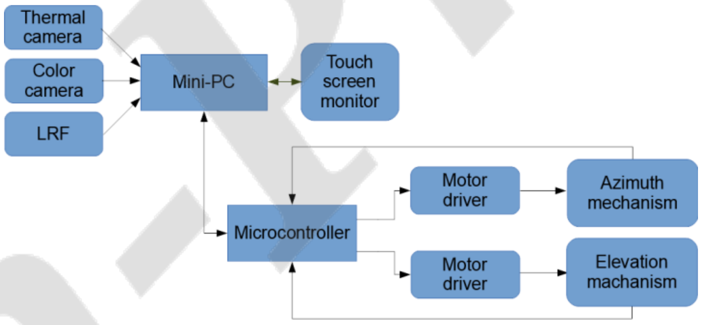


Figure 3. Telemonitoring system diagram of the two DOF robot arm

The telemonitoring system has been widely implemented in a variety of areas including the health field that Gyllensten has done for the treatment of patients with chronic heart failure [9]. Azimi et al. also had discussed a monitoring system for the elderly using the concept of the Internet of Things (IoT) [10]. Both of the telemonitoring systems above are more emphasis on the concept and how the telemonitoring system mechanism used but did not discuss the optimization to speed up the processing time, especially by using multi-core processors.

This paper describes the implementation of multi-thread method to optimize the robot telemonitoring system program. This system has been created using c programming language, OpenCV library, and Integrated Development Environment (IDE) Qt Creator.

2. Methods

Figure 2 is two Degrees Of Freedom (DOF) of robotic arms which have been developed in the Research Center for Electrical Power and Mechatronics, Indonesian Institute of Sciences (LIPI).

Microcontrollers are used to drive two DOF mechanisms, while mini-pc is used for complex data processing and requires fast processing times such as for image processing, stabilization systems, ballistics, and telemonitoring or telecontrol systems. Mini-pc specification used in this research is intel core processor i3-4160T, 8GB DDR3, 500GB HDD, Intel HD Graphics VGA, GbE NIC, and windows 7 professional.

Telemonitoring system diagram that has been created to monitor and control the robotic arm can be seen in Figure 3.

Mini-PCs read data from thermal cameras, color cameras, and LRF that are displayed to the user through a GUI-based touchscreen monitor. In addition, the user can send commands to the mini-pc through the GUI. The Mini-PC is connected to a microcontroller using a serial cable to transmit the azimuth and elevation angle coordinates and read the current azimuth and elevation positions. A microcontroller is an intermediary between a mini-pc and a motor driver that sends commands from a mini-pc to a motor driver and reads the current angle of the azimuth and elevation mechanisms.

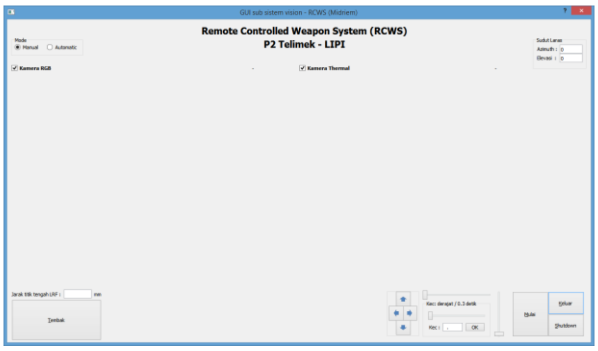


Figure 4. The telemonitoring system program based Graphical User Interface (GUI)

The application program of the telemonitoring system based Graphical User Interface (GUI) can be seen in Figure 4. The data displayed on the GUI i.e. thermal camera, color camera (RGB), azimuth and elevation angle, object distance, the choice of a manual or automatic mode in the movement of the robot arm, button to movements robot manually with speed setting which can be used in manual mode and "Tembak" button to shoot the object. If the automatic mode is selected, the robot arm will move to follow the direction of the selected object using tracking algorithms.Telemonitoring system flowchart can

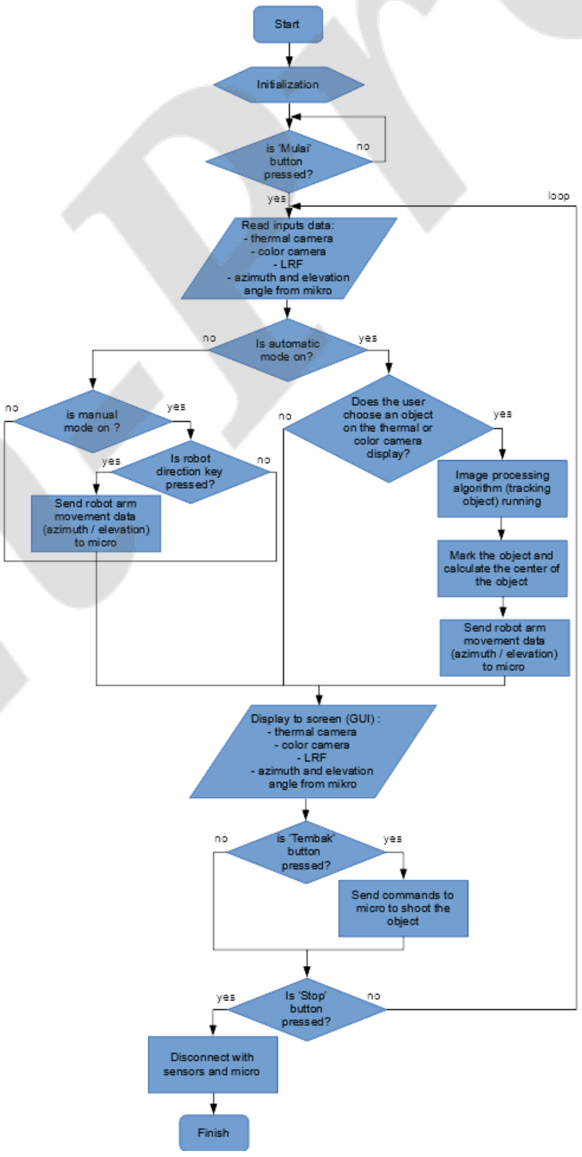

Start
Initialization
is 'Mulai' button pressed?
no
yes
loop
Read inputs data:
- thermal camera
- color camera
- LRF
- azimuth and elevation angle from mikro
Is automatic mode on?
no
yes
is manual mode on ?
no
yes
Is robot direction key pressed?
yes
no
Send robot arm movement data (azimuth / elevation) to micro
Does the user choose an object on the thermal or color camera display?
no
yes
Image processing algorithm (tracking object) running
Mark the object and calculate the center of the object
Send robot arm movement data (azimuth / elevation) to micro
Display to screen (GUI) :
- thermal camera
- color camera
- LRF
- azimuth and elevation angle from mikro
is 'Tembak' button pressed?
no
yes
Send commands to micro to shoot the object
Is 'Stop' button pressed?
yes
no
Disconnect with sensors and micro
Finish

Figure 5. Telemonitoring system flowchart

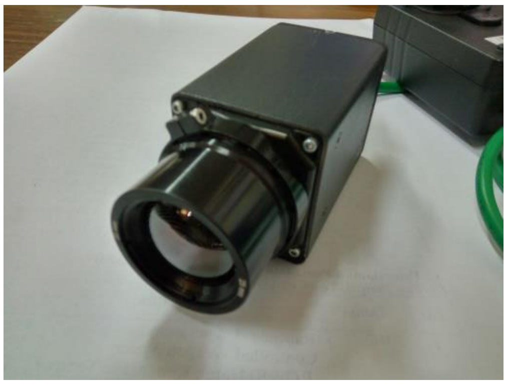


Figure 6. Thermal camera

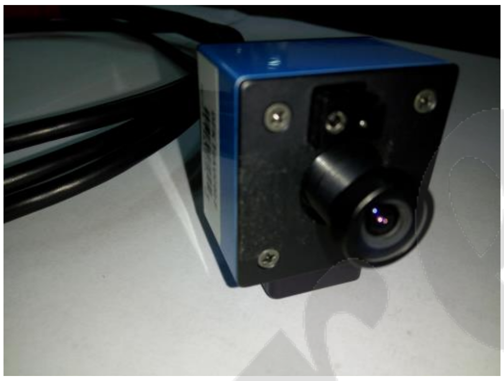


Figure 7. Color camera

be seen in Figure 5. The thermal camera used in this research is Flir A65 (f = 25mm, 7.5 Hz) which connected using Local Area Network (LAN) cable with anRJ45 connection, and the color camera is DFK 72AUC02-F connected via USB. The both of camera can be seen in Figure 6 and Figure 7.

One part that requires dynamic and high processing time on this telemonitoring system is the display of two cameras i.e. thermal camera and color camera with 640 x 480 pixel resolution with tracking algorithm used in real-time, it can be seen primarily in thermal camera display which less smooth, it can be seen in Figure 8. The computing process of thermal cameras and color cameras is done sequentially at any given interval using a timer. The object tracking algorithm used in this telemonitoring system is Tracking Learning Detection (TLD) [11].

The GUI display in Figure 8a is taken shortly after the bottle is moved to the front of the camera, while in Figure 8b is three seconds after it. In Figure 8a we can see the delay in the thermal camera shown by the absence of the bottle, whereas in Figure 8b has appeared. On the other hand, the color camera has a bottle showing, both in Figure 8a and Figure 8b, although there is still a little delay occurs.

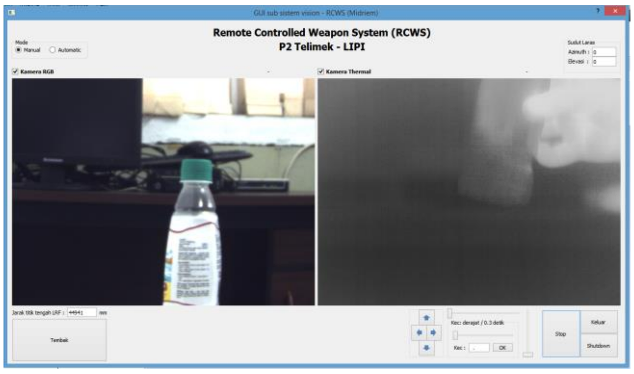
The sequential timing of sequential processing has been done at several time intervals to ensure real-time processing that can be seen in section three. From the test results can be seen that there is still a lag that causes the appearance of the camera on the GUI has not been smooth.

Based on this, two new threads have been added, first to process thermal camera, and second to process the color camera. Each thread is processed in parallel and separate, so it can handle the needs of a fast and dynamic processing. Two new threads implemented can be seen in Figure 9.

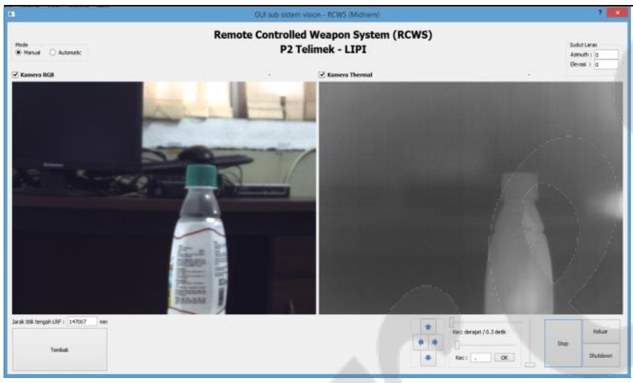
Illustrated of thread execution on this system on the processor cores can be seen in Figure 9. The first thread is used to process raw thermal camera data until display it in the GUI and the second thread is used to process raw color camera data until display it in the GUI. Core processor was used to executing this thread is set automatically, so no need to specify the program.

Flowchart of the new proposed system can be seen in Figure 10. In Figure 10 it can be seen that after the 'Mulai' button is pressed it will form two new threads that process color camera and thermal camera data continuously until the 'Stop' button is pressed. So the total thread was used is three.

If the user chooses automatic mode and chooses the object to be tracked either by the color camera or thermal camera, the tracking algorithm will work and send commands to the microcontroller to move the azimuth and elevation mechanisms to always point to the object, whereas if manual mode is selected, the tracking algorithm does not



(a)



(b)

Figure 8. GUI when running the program sequentially; (a) shortly after the bottle is moved to the front of the camera; (b) three seconds later.

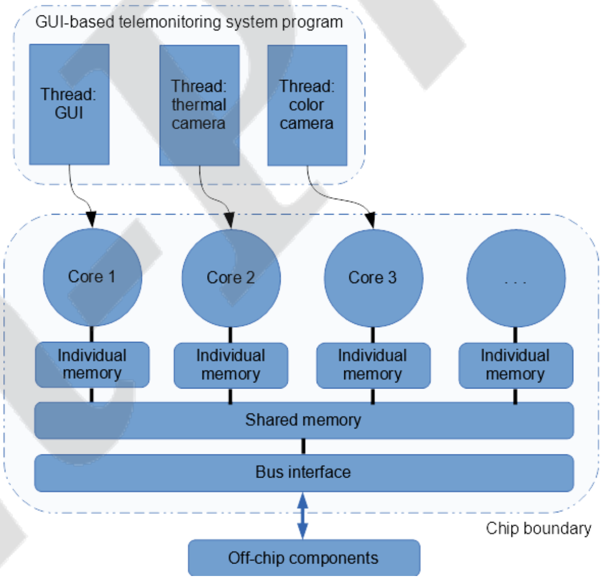


Figure 9. The implementation of multi-thread on telemonitoring system

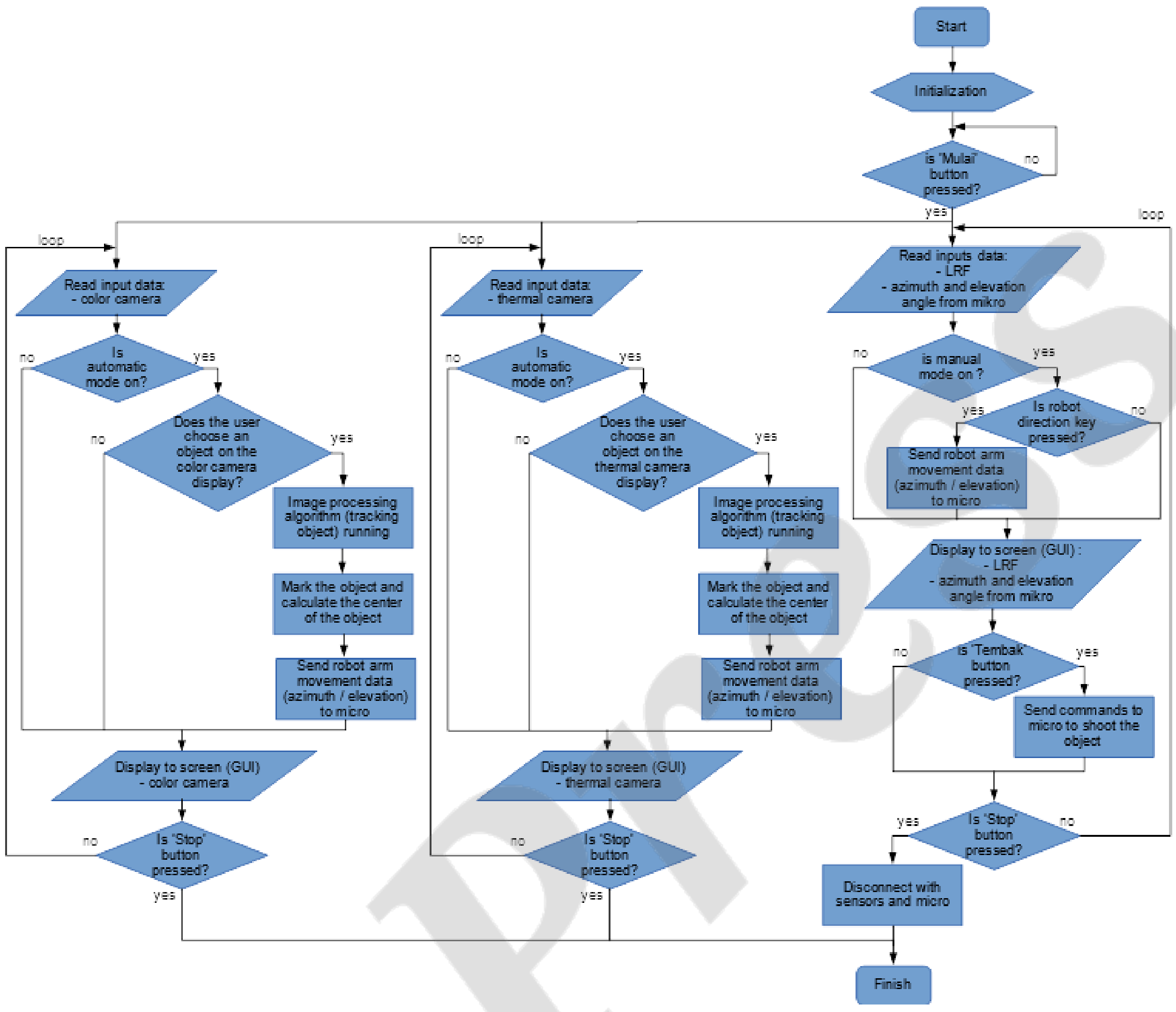
work, the system will only move the azimuth and elevation mechanisms according to the direction keys pressed by the user.

Thread had been implemented into an application program using Qt Creator IDE with `QtConcurrent` header. Source code had been made using c language is as follows.

```
//Header
#include <QtConcurrent/QtConcurrent>

// variables declaration
QFuture<void> thread_RGB;
QFuture<void> thread_Thermal;

//running thread
thread_RGB = run(this,
&Dialog::proses_RGB);
thread_Thermal = run(this,
&Dialog::proses_Thermal);
// Procedures to be executed on thread
void Dialog::proses_RGB() {
// Source code of color camera
// processing
}
void Dialog::proses_Thermal() {
// Source code of thermal camera
```



Start
Initialization
Is 'Mulai' button pressed?
no
yes
loop
Read input data: - color camera
Is automatic mode on?
no
yes
Does the user choose an object on the color camera display?
no
yes
Image processing algorithm (tracking object) running
Mark the object and calculate the center of the object
Send robot arm movement data (azimuth / elevation) to micro
Display to screen (GUI) - color camera
Is 'Stop' button pressed?
no
yes
loop
Read input data: - thermal camera
Is automatic mode on?
no
yes
Does the user choose an object on the thermal camera display?
no
yes
Image processing algorithm (tracking object) running
Mark the object and calculate the center of the object
Send robot arm movement data (azimuth / elevation) to micro
Display to screen (GUI) - thermal camera
Is 'Stop' button pressed?
no
yes
loop
Read inputs data: - LRF - azimuth and elevation angle from mikro
Is manual mode on ?
no
yes
Is robot direction key pressed?
yes
no
Send robot arm movement data (azimuth / elevation) to micro
Display to screen (GUI) : - LRF - azimuth and elevation angle from mikro
Is 'Tembak' button pressed?
no
yes
Send commands to micro to shoot the object
Is 'Stop' button pressed?
yes
no
Disconnect with sensors and micro
Finish

Figure 10. Flowchart telemonitoring system with added two threads

```
// processing
}
```

Two procedures have been created to process each of these cameras, and each procedure is processed on a different thread, `Dialog::proses_RGB()` procedure is processed on `thread_RGB`, while the `Dialog::proses_Thermal()` procedure is processed on `thread_Thermal`.

The processing time of a sequential program and with added two threads is calculated using `std::chrono::high_resolution_clock` [12] to calculate the actual processing time in a microsecond and `QTime` [13] to calculate Frame Per Second (FPS) of each process. Source code in c language using `std::chrono::high_resolution_clock` is as follows.

```
//Header
#include <chrono>

//Start calculation
auto start =
std::chrono::high_resolution_clock::n
ow();

//Read buffer of thermal or color
//camera & process the tracking
//algorithm

//Calculate elepsed time in us
auto elapsed =
std::chrono::high_resolution_clock::n
ow() - start;
long long microseconds =
std::chrono::duration_cast<std::chron
o::microseconds>(elapsed).count();
```

The source code in c language using `QTime` is as follows.

```
//Header
#include <QTime>

//variables declaration
QTime myTimer;
double fps;
double second;
int jml_counter = 0;
```

```
//Start calculation
myTimer.start();

//Source code of camera processing

//Calculate FPS
jml_counter++;
if (jml_counter >= 120) {
   second = myTimer.elapsed() / 1000;
   fps = jml_counter / second;
   myTimer.restart();
}
```

3. Results and Discussion

The sequential processing time was performed at 100 ms, 134 ms, 159 ms, and 300 ms time intervals as shown in Figure 11. In Figure 11 it can

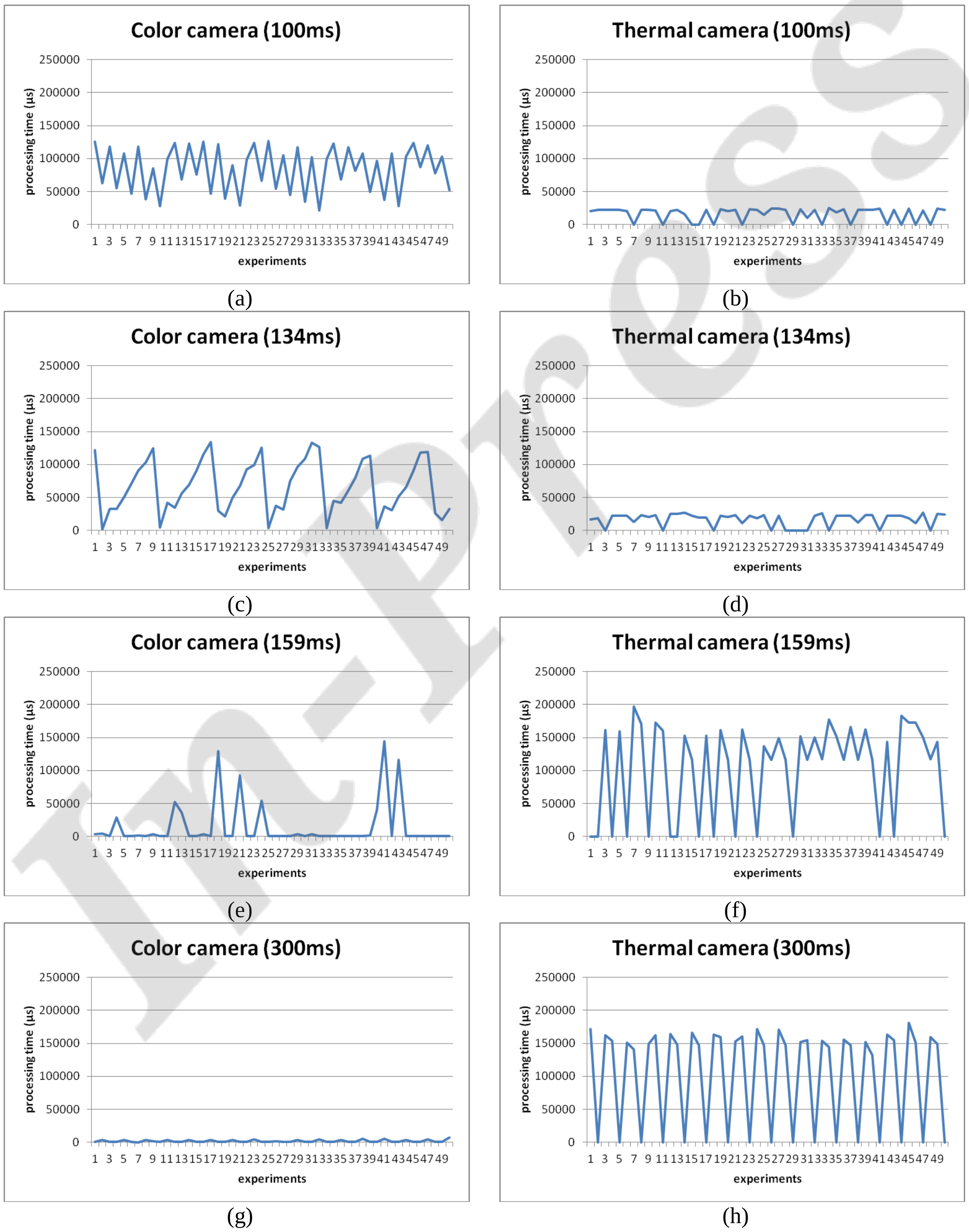


Figure 11. The sequential processing time of: (a) color camera 100 ms; (b) thermal camera 100 ms; (c) color camera 134 ms; (d) thermal camera 134 ms; (e) color camera 159 ms; (f) thermal camera 159 ms; (g) color camera 300 ms; (h) thermal camera 300 ms

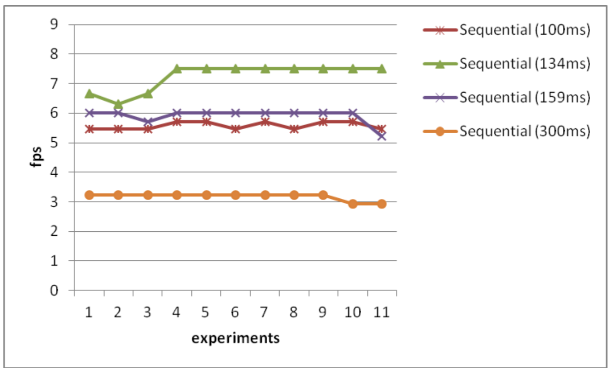
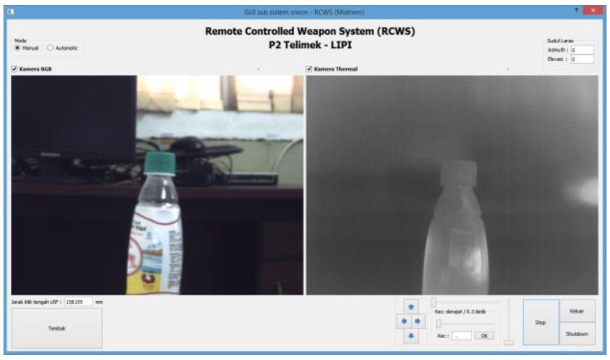
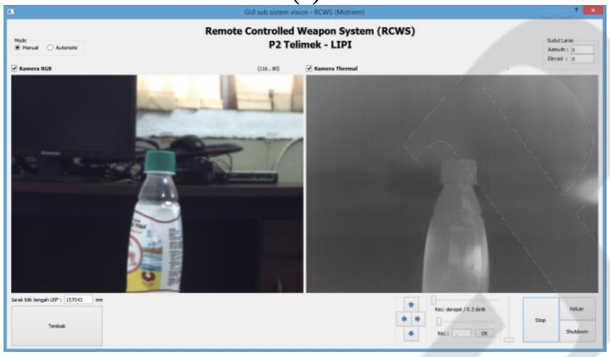


Figure 12. The sequential processing time at each interval in fps



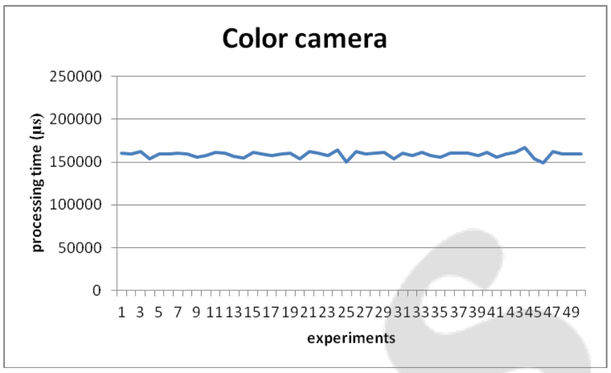
(a)



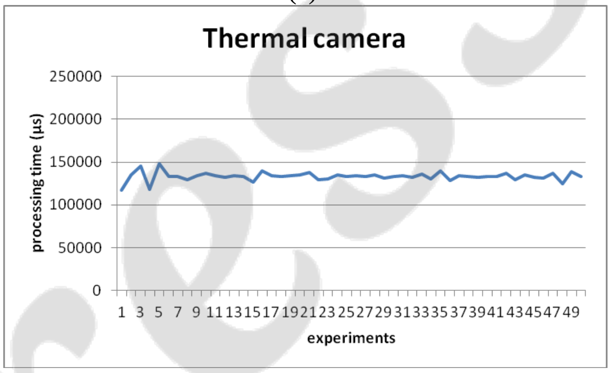
(b)

Figure 13. GUI with added two threads; (a) shortly after the bottle is moved to the front of the camera; (b) three seconds later.

be seen that the processing time of the thermal camera and color camera at 100 ms, 134 ms, 159 ms, and 300 ms intervals vary. This is related to the timer interval used and loads the process done, whether it is just reading the image buffer and displaying it, or accompanied by the processing of the tracking algorithm. Some value of the processing time of the thermal camera is 0 which means that the buffer is not processed at that time and causing the lag. Unlike the thermal camera, almost all value of the color cameras are more than 0, except at 300 ms intervals, it means that the amount of lag of the color camera is less than thermal camera. Based on the experimental results, it can be seen that all the timer intervals have not been able to process both cameras smooth and stable without any lag.



(a)



(b)

Figure 14. The parallel processing time of; (a) color camera; (b) thermal camera

Experiments have also been performed on PCs with Intel Core i7-4790, 8GB DDR3, 1 TB, Nvidia GeForce GTX 745, and Windows 8.1 as comparators. Based on experimental results, obtained the same conclusion, there are still lag on several experiments.

Fps generated from the sequential time of process at both 100 ms, 134 ms, 159 ms, and 300 ms intervals can be seen in Figure 12. In Figure 12 it can be seen that the highest fps time is obtained at intervals of 134 ms ie 7.5 fps, but it can not be used as a solution because the appearance is still not smooth because of the lag as shown in Figure 11.

The GUI display after added two threads can be seen in Figure 13. The GUI in Figure 13a is taken shortly after the bottle is moved to the front of the camera, while Figure 13b is three seconds later. In Figure 13a and Figure 13b, it can be seen that there is no delay like Figure 8, that means the display of thermal camera and the color camera has been running smooth.

The processing time of the thermal camera and color camera after each run parallel on a different thread can be seen in Figure 14. In Figure 14 it can be seen that the processing time of the color camera and thermal camera data is more stable than using a certain timer interval. In this way it can also accommodate dynamic processing, it can be seen from the absence of a zero processing time so that no lag occurs. The resultant fps on parallel

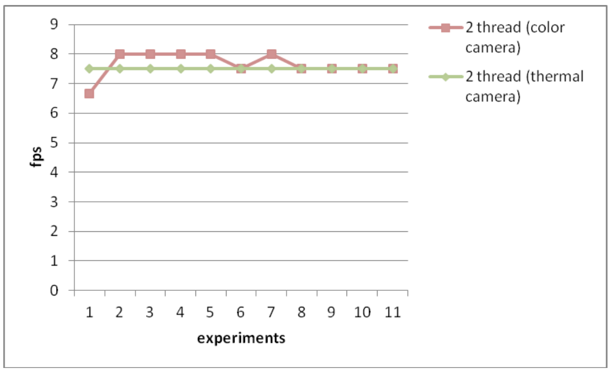


Figure 15. The parallel processing time in fps

processing can be seen in Figure 15. In Figure 15 it can be seen that the average fps generated on each camera's processing is more stable and is at the top level of 7.5 fps on the thermal camera and 7.5 - 8 fps in the color camera.

4. Conclusions

The multi-threaded method has been successfully implemented on two DOF robotic arm telemonitoring systems. The experimental results show that both the camera display with tracking algorithm used inside it after the process is separated using two threads can work smooth which can be seen visually, no lag in the processing time, and the average fps is more stable and is on higher level that is 7.5 fps on the thermal color and 7.5 - 8 fps on color camera. Based on this, it can be seen that the two threads added can improve the speed, and make it stable to make the display to be smooth without any delay like a sequential program.

Acknowledgments

Author would like to thanks to Research Center for Electrical Power and Mechatronics, Indonesian Institute of Sciences (LIPI) for the support of research funds, Mechatronics research group, especially to Aditya Sukma Nugraha, Hendri Maja Saputra, and Rifa Rahmayanti that have supported this research, and all those who have helped conducting this research.

References

[1] Intel Corporation, "*Intel® Core™ i7-6950X Processor Extreme Edition*", [Online]. Available: https://ark.intel.com/products/94456/Intel-Core-i7-6950X-Processor-Extreme-Edition-25M-Cache-up-to-3_50-GHz. [Accesed: 03 04 2017].

[2] Intel Corporation, "*Intel® Xeon® Processor E7-8890 v4*", [Online]. Available: https://ark.intel.com/products/93790/Intel-Xeon-Processor-E7-8890-v4-60M-Cache-2_20-GHz. [Accesed: 03 04 2017].

[3] Intel Corporation, "*Intel® Xeon® Processor E7-8894 v4*", [Online]. Available: https://ark.intel.com/products/96900/Intel-Xeon-Processor-E7-8894-v4-60M-Cache-2_40-GHz. [Accesed: 03 04 2017].

[4] B. Huynh, B. Vo dan V. Snasel, "*An efficient method for mining frequent sequential patterns using multi-Core processors*", Applied Intelligence, Vol. 46, No. 3, 2017, pp. 703-716.

[5] L. Cabaret, L. Lacassagne dan D. Etiemble, "*Parallel Light Speed Labeling: an efficient connected component algorithm for labeling and analysis on multi-core processors*", Journal of Real-Time Image Processing, No. Special Issue Paper, 2016, pp. 1-24.

[6] M. Mirdanies, A. S. Prihatmanto dan E. Rijanto, "Object Recognition System in Remote Controlled Weapon Station using SIFT and SURF Methods," Methatronics, Electrical Power, and Vehicular Technology, Vol. 4, No. 2, 2013, pp. 99-108.

[7] R. Zhu, L.-h. Qin, J.-l. Zhou dan H. Zheng, "*Using multi-threads to hide deduplication I/O latency with low synchronization overhead*", Journal of Central South University, Vol. 20, No. 6, 2013, pp. 1582-1591.

[8] M. Sikora dan I. Mateljan, "*A method for speeding up beam-tracing simulation using thread-level parallelization*", Engineering with Computers, Vol. 30, No. 4, 2014, pp. 679-688.

[9] I. C. Gyllensten, A. Crudall-Goode, R. M. Aarts dan K. M. Goode, "*Simulated case management of home telemonitoring to assess the impact of different alert algorithms on work-load and clinical decisions*", BMC Medical Informatics and Decision Making, Vol. 17, 2017, pp. 1-11.

[10] I. Azimi, A. M. Rahmani, P. Liljeberg dan H. Tenhunen, "*Internet of things for remote elderly monitoring: a study from user-centered perspective*", Journal of Ambient Intelligence and Humanized Computing, Vol. 8, No. 2, 2017, pp. 273-289.

[11] Z. Kalal, K. Mikolajczyk dan J. Matas, "*Tracking-learning-detection*", IEEE transactions on pattern analysis and machine intelligence, Vol. 34, No. 7, 2012, pp. 1409-1422.

[12] cppreference.com, "*C++ reference*", 2017. [Online]. Available: http://en.cppreference.com/w/cpp/chrono/high_resolution_clock. [Accesed: 13 04 2017].

[13] The Qt Company, "*Qt*", 2017. [Online]. Available: http://doc.qt.io/qt-5/qtime.html. [Accesed: 13 04 2017].