Options and Pitfalls in Embedded Systems Development for Intelligent Humanoid Robots

Jacky Baltes[1], Kuo-Yang Tu[2], and John Anderson[1]

[1] Jacky Baltes, University of Manitoba, Winnipeg, MB, Canada R3T 2N2
j.baltes@cs.umanitoba.ca
http://www.cs.umanitoba.ca/jacky
[2] National Kaohsiung First University of Science and Technology,
811 Kaohsiung, Taiwan

Abstract. This paper describes the most popular options that are available developers of intelligent humanoid robots and their advantages and disadvantages. There has never been a wider range of affordable and practical solutions for the developers of intelligent humanoid robots. This paper lists the suitability of the most common options such as microcontrollers, ARM based embedded boards, and x86 based small PCs and how well the meet different design constraints. Using an example from low level vision processing, the paper highlights common pitfalls when including these more complex embedded systems in their robot.

1 Introduction

In recent years, there have been rapid improvements in computer technology in general, and in the embedded systems space in particular. The so-called Moore's law is the observation that the number of transistors on a die follows an exponential growth curve and doubles approximately every two years. Even though the end of this trend has been predicted many times, it has continued for the last 40 years.

The speed-up has not just benefitted workstations, PCs, and super computers but has also led to increased performance of embedded systems platforms. Section 2 givse an introduction to the current landscape in embedded systems.

2 Embedded Systems Hardware

There are many types of robots with a waste array of intended uses and applications. To build a robot, its designer has to balance many different factors when choosing an embedded system. For example, the designer has to consider:

- the size and weight,
- the processing power (e.g., ARM 700Mhz or Intel Atom 1.2GHz),
- the amount of RAM (e.g., 256MB RAM),

K. Omar et al. (Eds.): FIRA 2013, CCIS 376, pp. 77–89, 2013.

- the amount and type of secondary storage (e.g., Flash RAM),
- the available IO functions (e.g., GPIO, ADC, DMA, USB, and interrupts)
- the display (e.g., LEDs, LCDs, VGA),
- the communication support (e.g., serial ports, ethernet, Bluettooth, WIFI)
- the power consumption,
- the durability and packaging,
- the available operating systems and their support,
- the programming language support and associated costs (e.g., C or ASM),
- the availability of software libraries for common tasks and associated costs,
- the restrictions imposed by software and hardware libraries (e.g., proprietary codec or open source),
- and finally the cost of the system.

Note that many of these design goals are contradictory. For example, a more powerful processor and more RAM will usually result in a higher cost. Similarily, lots of IO and display will increase the size and weight of the embedded system.

Based on these and possibly other factors (e.g., familiarity with a manufacturer, developed in house), the designer has to find an embedded system that maximzes the quality of the overall system, that is, the designer has to find the sweet spot in this forrest of constraints.

A comprehensive list of embedded systems used in intelligent robotics would be much too large for this paper. Therefore, we will limit ourselves to some of the most popular embedded systems. Since we are always on the lookout for better embedded systems for our robot, we encourage readers to inform us of new embedded boards that are uniquely suitable for the development of intelligent humanoid robots.

The following subsections provide some detail about the embedded systems that were used in the Autonomous Agents Lab at the University of Manitoba in Winnipeg, Manitoba and the System Identification and Control Lab at the National Kaohsiung First University of Science and Technology in Kaohsiung, Taiwan. The described systems give a good overview of the landscape of embedded systems available to the robotics researcher or hobbyist in 2013.

Furthermore, it must be noted that the lines separating the different classes described in the subsection below are fluid and constantly changing. For example, it is possible to find ARM based boards that have 4GB of RAM or x86 based boards with GPIOs. The subsections below discuss typical example of the devices found in the different subsections.

2.1 AVR ATMega 128

The AVR ATMega 128 is an extremely popular micro-controller that is powerful enough to control simple robots. The ATMega128 is an 8-bit micro-controller that runs at 16MHz. It features a full range of IO options including digital general purpose input output pins (GPIO), analog to digital conversion channels (ADC), an I2C bus, and several pulse width modulation generators (PWM), that are useful in controlling DC motors directly via an H-Bridge.

Many companies use AVR ATMega controllers in their products. For example, the Robotis company, a large Korean robotics company, use them as controllers for their intelligent servos and as robot controllers [13]. Similarly, the extremely popular Arduino series of embedded systems are based on AVR ATMega processors [2]. See Fig. 1 for an example of a typical AVR ATMega 128 based robot developed by the Taibotics company in Taiwan [10].

The communication channels of the AVR ATMega series is limited to serial communication, which can be extended to bluetooth using simple RFCOMM Bluetooth modules for wireless communication. Since fewer and fewer laptops and PCs still include serial ports, the newer generation of these types of processors include USB client support, which allows the designer to connect the embedded system to the USB port of a laptop. Note that this however does not allow the embedded system to act as a USB host. Thus it is not just a matter of adding a USB Webcam to the embedded system to provide vision for the robot.

The hardware of the AVR ATMega series is not substantially superior to that of its competitors; the greatest strength lies in the availability of high performance free and open source programming environments (AVR-GCC) and support libraries [8], [14]. There are also many free and open source operating systems kernels available for this device. For example, we developed our own real-time kernel called Freezer OS [4], which has been used by some other robotics teams world wide. Another advantage of the AVR series is the ease with which it can be deployed in a system, since it provides in system programmability via its built in boot loader protocol.

The biggest disadvantage of the AVR ATMega 128 is its small amount of built-in RAM (4 KB) and its slow processing speed. This means that it is not practical to implement vision or similar sensors without: (a) either limiting the robot to extremely small images or (b) requiring the addition of external RAM to the embedded systems design. The first option results in poor vision for the robots, but it is possible as demonstrated by the FUmanoids team used such an approach in their RoboCup team [5]. The latter option increases the cost of the design and does not overcome the problem of the slow processor speed, which means it is rarely used.

Similarly, the slow processor speed and small RAM also makes it impractical to add WIFI or Ethernet to the embedded system.

2.2 ARM Based Embedded Systems

Given the restrictions when designing a robotic system that uses vision, such as a fully automated humanoid robot, robotics designers choose more powerful ARM based embedded systems as the brain of their robots. These boards also provide a more complete operating system. For example, they often have available stripped down or nowadays more and more a full version of the Linux kernel with associated user land programs and libraries. Many of the devices also include a USB host interface.

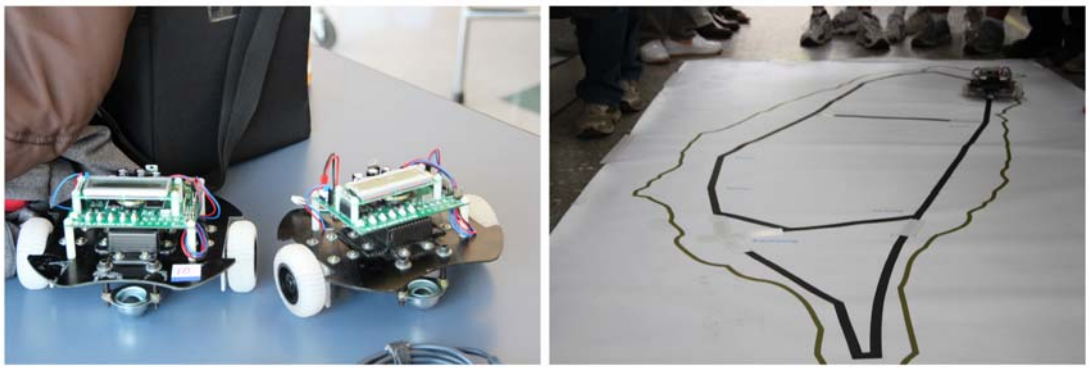

Fig. 1. A simple educational robot based on a single AVR ATMega128 micro-controller with three light sensors (Left) and a line tracer travelling around Taiwan (Right).

A typical example of an ARM based device suitable for control of a humanoid robot is the Raspberry Pi [7]. The Raspberry Pi, a credit card sized single board computer developed by the Raspberry Pi Foundation includes a 700MHz ARM processor and a graphics GPU. Due to its low price of less than $30 USD, the initial release of the Raspberry PI on February 2012 was a great success with units being sold out within hours. A similar device is the Beagleboard [6], which is more costly, but has better open source support and requires fewer proprietary and patent encumbered device drivers to power its graphics processing unit (GPU). The amount of RAM on those devices is about 256MB to 512MB.

2.3 x86 Based Nano PCs

In recent years, some x86 based very small PCs have become available to the robotics developer. These systems are similar to the ARM based embedded systems described in subsection 2.2, but usually have more RAM (1 GB). The IO options on these devices are the ones one would expect from a PC including Ethernet, WIFI, SATA disks, multiple USB host ports, and VGA.

A typical example of an x86 based Nano PC is the FitPC from Compulab in Italy [9]. It has a 1.6 GHz Atom processor with 1 GB of RAM. The system includes a 4GB Nand Flash for mass storage. It includes one USB port, WIFI, and an Ethernet port. Output is provided by a HDMI output port. The FitPC is used in the popular DARwIn OP robot [12], currently the top of the line humanoid robot sold by Robotis.

The advantage for a developer is that the devices can function as a full PC and can run standard Linux distrubutions such as Ubuntu or Fedora. This means that the developer's software environment (the build environment) and the robot's software environment (the target environment) are identical. These removes issues in cross compilation and staging of the environment. For example, the target environment may only include an older version of a necessary library. Furthermore, the fact that the x86 platform is by far the most popular Linux environments means that software is usually more stable on these platforms.

A disadvantage of many of these boards is that these board only provide common PC IO interfaces, but do not include GPIO or ADC pins. Therefore adding extra sensors or actuators (e.g., balancing gyroscope or finger servo) is difficult or may be even impossible given the high integration of these devices. In these cases, the designer's only option is to create a common PC IO (e.g., USB client) based interface for the sensor or actuator, which usually increases cost and decreases throughput for the device.

The Roboard RB-110 is a hybrid device in the sense that it uses an x86 based CPU and many common PC interfaces, but also about 40 pins that can act as GPIO, ADC, or PWM generators [1]. The RB-110 board is a small embedded system being manufactured by DMP Inc. from Taiwan. It includes a 1GHz x86 Vortex CPU (similar to the Intel 486 CPU) with 256MB Ram. It includesEthernet, two serial ports, four USB host ports, . A VGA display card or a WIFI card can be added to the board via a mini PCI port on the board. Mass storage for the OS is provided by a micro SD card.

2.4 Smartphones

In 2006, the team UofM Humanoids was the first team to use commercially available smartphones (Nokia N6600) as the brains of their humanoid robots (see Fig. 2. The advantage of using a mobile phone was that it provides good computing power, large RAM, a camera, display, and buttons at a very low cost. This is due to the fact that mobile phones are commodity products that are produced in the millions reducing the cost per unit. The embedded systems described in the previous subsections are produced in small runs and thus their cost per unit is much higher. Furthermore, the fact that companies produce new mobile phones are introduced almost every six month meant that there is a big market for used mobile phones.

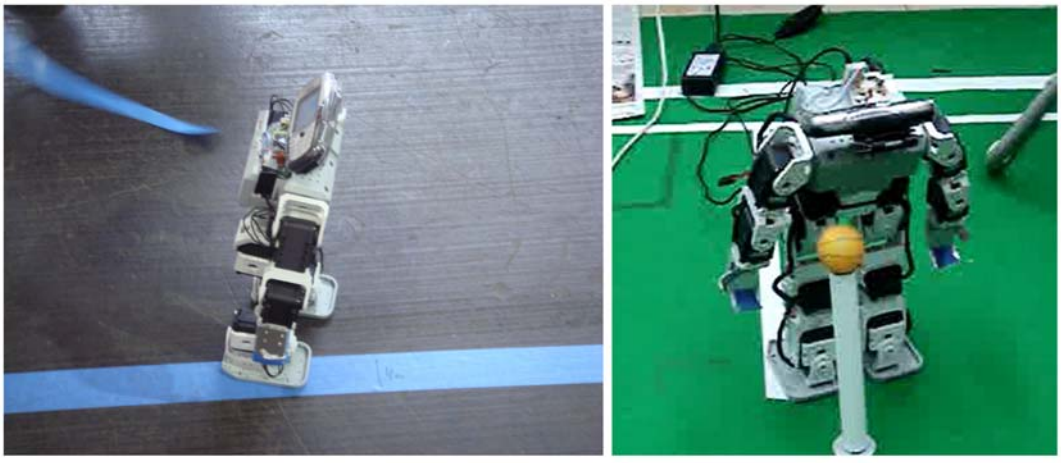

Fig. 2. Our humanoid robots using mobile phones as embedded systems. The mechanics are based on the Bioloid Kit from Robotis, Korea. Communication to the sevo motors is via a custom infrared board.

More recently, some teams have used IPhones or Android phones as cheap embedded systems for their humanoid robots. One worrying trend is that developers are being more and more limited by the manufacturers. Whereas in 2006, free development tools for the Symbian OS were made available by Nokia, later versions required an Internet connection to sign the binaries, so that they can be downloaded to the robots. In 2013, Apple and Windows Mobile require developer accounts, which cost about $100 USD per year and application can only be legally downloaded via the company's Appstore.

One disadvantage of the phones is that their IO is extremely limited. Luckily, the Nokia phones that we used included the almost obsolete IRDA standard for infrared communication. We designed a small IRDA board to communicate with the AVR ATMega based micro-controller on the robot. The situation is slightly better for Android based phones. Google and other companies manufacture small IO boards that allow a developer to connect the smartphone to a micro-controller via the USB bus.

Another is that the camera is mounted in the phone and thus fixed with respect to the robot. This means that the whole robot has to turn to look sideways or the robot has to move its torso to look down, which greatly slows down the robot. Many applications can benefit from a pan and tilt camera, but the whole smartphone is quite heavy and bulky, so it would require large motors to move quickly and accurately.

3 Pitfalls of Embedded Systems Development

Whereas the availability of powerful embedded systems is a clear boon to robotics enthusiasts, the additional complexity of these devices has made the design of intelligent humanoid robots more challenging. The complexity increases in both hardware and software.

Computer vision is a common bottleneck on our robots. It is a difficult task where large amounts of information need to be captured, processed, and then acted on. Even seemingly simple tasks such as tracking a coloured object (e.g., the ball in a soccer match) require non trivial colour space conversions and geometric transformations. This is due to the fact that the pixel values greatly change with the brightness of the field. See Fig. 3 for the output of our vision system.

As this initial segmentation step is extremely important, we spent a lot of time on trying to optimize the performance of this vision process. We target a minimum frame rate of 10Hz or 100 milliseconds per image for our vision system, since otherwise the robot will miss fast moving soccer balls or other important features in the environment. However, any speed up in the vision processing is beneficial to the other components of the system such as task planning or navigation of the robot.

No it is no surprised that over the years we have developed fast and efficient algorithms, but every time we move to a new embedded platform, we try to optimize the parameters of our system to make sure that it works as efficiently

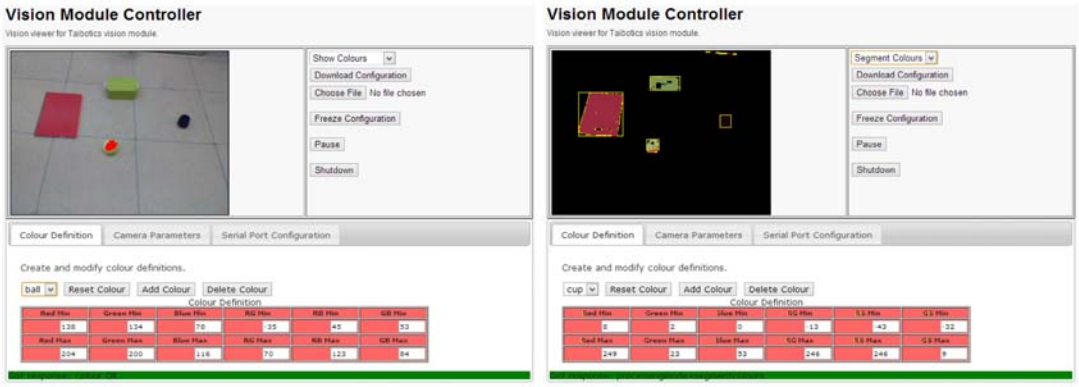

Fig. 3. Output of our Vision System. The left image shows the raw image and the selected colour. The right image shows the output of our segmentation algorithm.

as possible. In fact, the main point of this paper came about because of our experiences duing our optimization attempts on the RB-110 board as described in the subsection above.

3.1 Optimization as the Root of All Evil

Optimizing has always been something of a black art and should be used with care. Donald Knuth, one of the most prolific researchers in computer science, is often misquoted: "*Premature optimization is the cause of all evil.*" [11]. This is seen by some developers as an indication to avoid optimization completely.

However, Knuth's complete quote is:

> We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified

So Knuth does not say optimization is evil, but rather that before optimizing a piece of code, the developer has to make sure that the optimized code is part of the generally small piece of code, whose optimization can lead to much improved performance for the complete system. The authors would add to this that the code also first needs to be checked for correctness, since optimized buggy code will produce the same incorrect result as buggy code, just faster.

Unfortunately, there are still a lot of rules of thumbs that are used by developers which are not applicable to modern CPUs. One example is the technique of optimizing code by reducing the number of instructions or reducing the complexity of the instructions (e.g., remove an integer multiply with a shift operation). Modern processors have complex out of order execution pipelines that can execute several instructions in parallel. Furthermore, complex arithmetic logic units (ALUs) can nowadays execute floating point instructions almost as fast as simple integer add instructions.

Instead, many bottlenecks on modern CPUs are due to inefficiencies in using the memory cache hierarchy. Since the speed of the modern CPUs has increased much more quickly than that of RAM, modern CPUs include multuple levels of caches to try and amortize the cost of memory accesses and of prefetching data from memory that is likely to be used in the near future. Being unaware of the impacts of the caches on the performance can lead to poor performance. For example, the frame rate of vision system dropped from 10 frames per second to less than one frame per second when we switched the order in which we processed the image from row major to column major order.

So we usually spend quite a bit of time identifying the bottlenecks in our system before attempting optimization. Workstation processors include better and better support for profiling of code to find bottlenecks and the same tools are now increasingly available for embedded systems.

3.2 Modern Robotics System Block Diagram

A typical vision system for a humanoid robot includes several hardware and software components, some of them that are outside of the control of the designer. Figure 4 shows the major hardware and software components of our vision system.

In our vision system, the application requests frames from the device driver of the webcam. The request is forwarded to the USB host controller, which then starts a DMA transfer to transfer the image to RAM. Note that this DMA transfer is supported by the hardware and does not require much work from the processor. The task manager of the OS schedules the various tasks and wait queues (e.g., processes that are waiting for the DMA transfer of the image to conclude). The memory hierachy is controlled by the OS memory manager. The application then reads and processes the frame in chunks that depend on the characteristics and amount of cache memory available. The vision system itself also often has to write a frame back into RAM for further processing or display.

3.3 Influence of Image Size on Processing Speed

Most modern cameras allow the user to select the resolution of the captured image, since not all applications require the maximum resolution images (e.g., photography vs. video chat). Common image resolutions are 80x60, 160x120, 320x240, 355x288, and 640x480. Since a larger image results in more accurate vision, but also generally requires more processing time and more memory space, this decision is important in the design a vision system. Since speed is important, most of our vision algorithms only do one pass over the image, which results in a worst case complexity of $O(n)$ for our algorithms. As the image size increases, you therefore expect an increase in runtime that roughly follows the number of pixels in the image. So for example, a 320x240 resolution image should take about four time as long as processing a 160x120 resolution image.

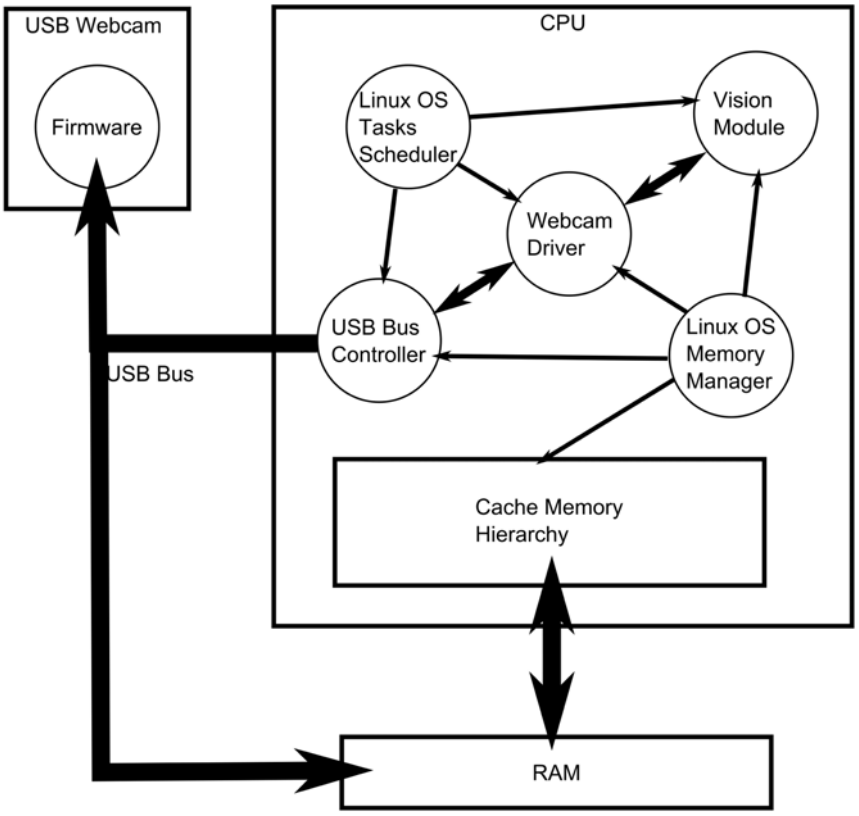

Fig. 4. Block Diagram of a modern embedded system. Requests by the vision system are serviced by the device driver and data is being sent over the USB bus by the USB bus controller. Tasks and memory are managed by the OS.

In the case of soccer playing robots, it is more important to react to quickly react to a ball that is close to the robot rather than to determine the exact position of the ball when it is far away from the robot. So we usually use an 80x60 resolution for our images in this domain.

We conducted the following experiments to measure the performance of our vision system on the RB-110 board. We extended our vision system to measure the time that it took from the initial rquest until the device driver signaled that the image was received. In order to utilize as much of the possible CPU time as possible, our vision system incudes a separate capture thread that continuously requests images from the camera and stores them in main memory. Whenever the vision system requests a new frame, the latest frame in memory is made available to the application. In parallel the new frame is downloaded into main memory via the hardware. The vision system uses colour to identify various blobs in the image and calculates their average colour, size, bounding box, and aspect ratios.It also creates a new frame in main memory which shows the detected objects and then transmits this frame via ethernet to a separate viewer. All the experiments described in this paper are done using the widely available

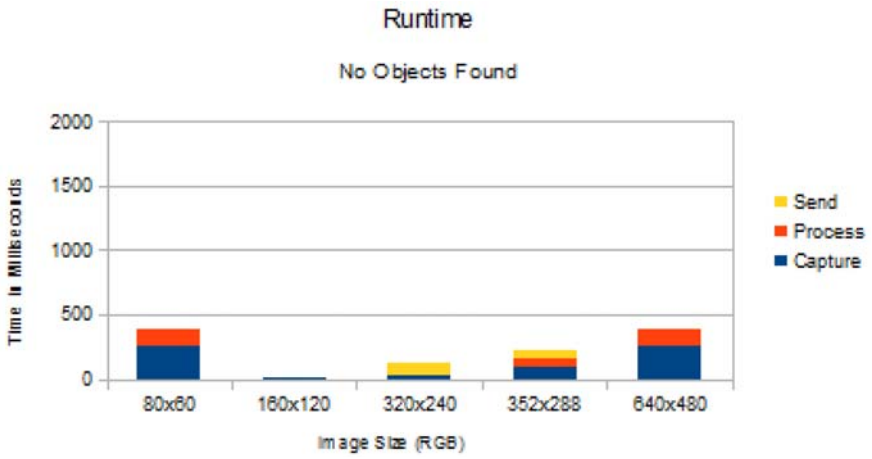

Fig. 5. Impact of Image Resolution on Processing Speed. There were no trackable objects in view. Note the anomaly of the slow 80x60 processing time.

RGB colour space. Similar results can be obtained by using other colour spaces such as HSI or YUV.

We noticed poor performance of our vision system using on the RB-110 board in our first trials. After investigating the situation more closely, we found that the problem was with the firmware of the webcam. As one can see from the chart, the time to capture an 80x60 resolution image is almost as high as that for a 640x480 camera image. We investigated the situation further and found that the firmware of the webcam did not properly support this resolution and did not allow DMA transfers of the image. Therefore, the time was much slower for the download and processing of the image than one would expect based on the image resolution. We therefore discarded 80x60 as a practical image size for our application.

Fig. 6 shows the peformance of the vision system for various image resolutions when tracking a single object. After eliminating the 80x60 resolution from the tests, the performance of the vision system is roughly what one would expect. Processing a 640x480 resolution image is approximately 16 times slower than a 160x120 resolution image and about four times slower than a 320x240 resolution image.

Figure 7 shows the performance of the vision system when tracking four large objects that contain many pixels each. In this case, the input and output frame must be visited several times to create the output image. So efficient use of the cache memory becomes important. The 640x480 resolution image requires many more cache reloads and performs poorly in this case. When trying to process such large resolution images on an embedded system with a small amount of cache memory, it would be more efficient to break the complete image into smaller cache aligned subimages.

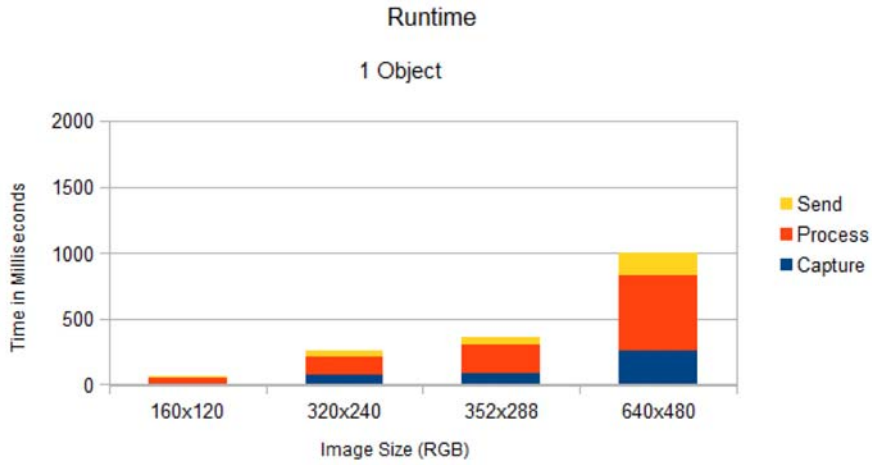

Fig. 6. Impact of Image Resolution on Processing Speed. The system was tracking a ball object.

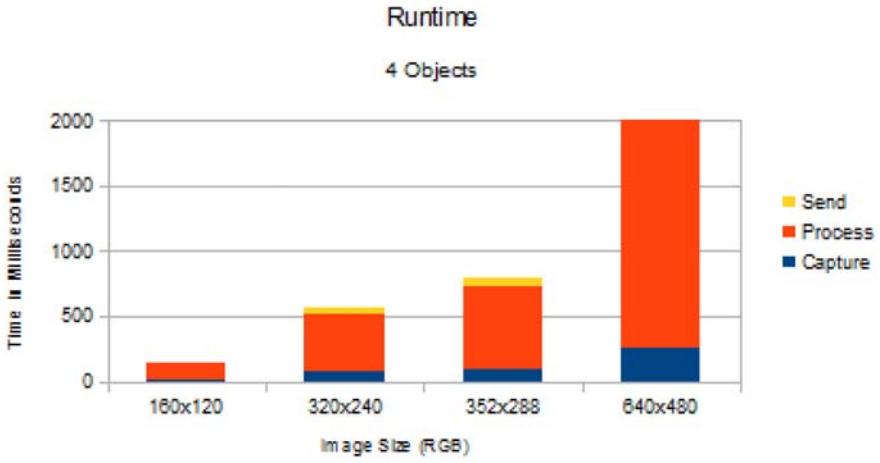

Fig. 7. Impact of Image Resolution on Framerate. The system was tracking four objects.

4 Conclusion

The development of intelligent robotics has seen rapid improvements in recent years. The rules of the HuroCup competition, the most challenging competition for humanoid robots, must be revised yearly to compensate for the improved performance of the robots [3]. For example, in 2007, the team KAIST from Korea won the inaugural marathon run over a distance of 42m in 37 minutes. In 2009, we decided to increase the length of the maraton to 82m, since the team aiRobot from NKFUST in Taiwan won the marathon over a distance of 42m in 7 minutes. In 2012, the marathon event over a distance of 82m was won by team UK Plymouth from the U.K. in 11 minutes. To keep the competition challenging for teams, the marathon distance was increased to 120m for the 2013

competition. Undoubtedly fueled by cheaper and better actuators, sensors, and mechanical structures of the robots, a significant portion of this improvement is also due to more powerful embedded systems. For example, this allows the teams to use better vision and balancing algorithms.

The authors feel that one increasingly worrying trend is that manufacturers and developers make it harder for other people to develop software for their general purpose computing devices. Steve Job's first commercially successful computer, the Apple II, included a programming language (Basic) and complete listing of the ROM of the computer. So everyone could write their own software and make use of the device in their own way. In fact, it would have been ridiculous to suggest that a user is not allowed to write and run their own program on a device that they legally purchased. Contrast this to Apple's latest devices, the IPhone or IPad. A user must join the Appstore for a yearly fee to legally develop and run software on their own IPhone or IPad. Also, a developer can not distribute his program to others unless the program was accepted by Apple and included in the Appstore. We used Apple in this case as an example, since it introduced the application store development model for computers, but now all other major companies (e.g., Microsoft Windows, Google Android, Microsoft Mobile) follow a similar strategy.

However, many great societal advances in recent years (e.g., the Internet, personalized medicine, robotic automation) used hardware and software in novel and unpredictable ways and restricting peoples' ability to develop software for their own needs is counter productive to improving our society. By making it harder and harder to develop your own solutions for your own needs, we may prevent the next Jobs and Wozniak from being able to develop solutions and thus kickstart the next Internet revolution.

References

1. RoBoard – The Heart of Robotics (March 2013), http://www.roboard.com/RB-110.htm
2. Arduino: Arduino homepage (March 2013), http://www.arduino.cc/
3. Baltes, J.: Hurocup facebook group (March 2013), http://www.facebook.com/group/hurocup
4. Baltes, J., Iverach-Brereton, C., Cheng, C.T., Anderson, J.: Threaded C and freezer OS. In: Li, T.-H.S., et al. (eds.) FIRA 2011. CCIS, vol. 212, pp. 170–177. Springer, Heidelberg (2011)
5. Free University Berlin: Fumanoid homepage (March 2013), https://www.fumanoids.de
6. Beagleboard Foundation: Beagleboard homepage (March 2013), http://www.beagleboard.org/
7. Raspberry Pi Foundation: Raspberry pi homepage (March 2013), http://www.raspberrypi.org/
8. FSF: Avr-gcc homepage (March 2013), http://gcc.gnu.org/wiki/avr-gcc
9. Compulab Inc.: Fit pc homepage (March 2013), http://www.fit-pc.com/web/
10. Taibotics Inc.: Taibotics homepage (March 2013), http://www.taibotics.com/

11. Knuth, D.E.: Structured Programming with go to Statements. ACM Computing Surveys 6(4), 261–301 (1974), http://portal.acm.org/citation.cfm?doid=356635.356640
12. Robotis: Darwin robotsource (March 2013), http://www.robotsource.org/xe/Introduction
13. Robotis: Robotis homepage (March 2013), http://www.robotis.com/
14. AVR Libc Team: Avr libc homepage. http://www.nongnu.org/avr-libc/ (March 2013)