An Open-Source Framework for Testing Tracking Devices using Lego Mindstorms

Julien Jomier[a], Luis Ibanez[a], Andinet Enquobahrie[a], Danielle Pace[b] and Kevin Cleary[c]
[a]Kitware Inc, 28 Corporate Drive, Clifton Park, NY, USA;
[b]Imaging Research Laboratories, Robarts Research Institute, London, ON, Canada;
[c]Imaging Science and Information Systems, Georgetown University, Washington, DC, USA

ABSTRACT

In this paper, we present an open-source framework for testing tracking devices in surgical navigation applications. At the core of image-guided intervention systems is the tracking interface that handles communication with the tracking device and gathers tracking information. Given that the correctness of tracking information is critical for protecting patient safety and for ensuring the successful execution of an intervention, the tracking software component needs to be thoroughly tested on a regular basis. Furthermore, with widespread use of extreme programming methodology that emphasizes continuous and incremental testing of application components, testing design becomes critical. While it is easy to automate most of the testing process, it is often more difficult to test components that require manual intervention such as tracking device.
Our framework consists of a robotic arm built from a set of Lego Mindstorms™ and an open-source toolkit written in C++ to control the robot movements and assess the accuracy of the tracking devices. The application program interface (API) is cross-platform and runs on Windows, Linux and MacOS.
We applied this framework for the continuous testing of the Image-Guided Surgery Toolkit (IGSTK), an open-source toolkit for image-guided surgery and shown that regression testing on tracking devices can be performed at low cost and improve significantly the quality of the software.

Keywords: Regression testing, Algorithm validation, Surgical navigation, Tracking devices.

1. INTRODUCTION

The testing of tracking devices as always been a challenge, especially in the context of the extreme programming software process[1] where testing is an important component of the development process. Moreover, software using tracking devices as input of the system often make use of simulation file in order to achieve proper testing coverage. However to improved robustness of the software, real tracking devices are needed for testing.

We present an open-source, cross-platform application program interface (API) for the automatic testing of surgical tracking devices. Combined with a Lego Mindstorms robot our framework provides a low cost testing framework for software and hardware developers. From a Lego Mindstorms set we build a 3-DOF robot arm. We then program, via the provided API, the robot to move a tracker tool given a specified pattern and record the resulting position and orientation of the tracker, providing a testing baseline. Every night, the robot movements are compared with the baseline and if a significant difference is found, an error is reported. More complex testing patterns can also be programmed since tracking devices are, most of the time, part of a longer processing pipeline. For instance, real tracking input associated with a real-time image-processing algorithm, such as registration, during surgery, allows for better testing of the overall system in a real world context.

Medical Imaging 2009: Visualization, Image-Guided Procedures, and Modeling,
edited by Michael I. Miga, Kenneth H. Wong, Proc. of SPIE Vol. 7261, 72612S

Next we describe the importance of software validation in the development of medical applications, and then we explain how our framework is implemented. Finally we demonstrate an application of the system for image-guided surgery applications.

2. SOFTWARE VALIDATION

Validation is an important part of any software development and can present some challenges. Nowadays, different approaches to software development have been adopted from the waterfall approach which pushes the testing phase at the end of the development cycle to the extreme programming approach which uses short development cycles and requires continuous testing. In this paper we are focusing on the extreme programming approach and we demonstrate how validation and testing play an important role in the quality of the final software.

2.1 Regression testing

By definition, any software must be continuously tested during its development due to the complex nature of the system. Usually, the last stage, before the release, of the software development is dedicated to validation and testing. However, in the context of extreme programming, developers write tests at the same time they write source code, which means that as soon as the code is committed in the source repository - also known as concurrent version system (CVS) or Subversion (SVN) - the code is automatically tested and, therefore, errors can be fixed more quickly. Several open-source projects have adopted this continuous development cycle as shown in figure 1. First developers add features or change code and commit it into the source repository, when the code is committed several platforms check out the code, compile it and run the different validation tests. The results of the validation are then submitted, through an online interface, back to the developers who keep fixing the code until the validation passes.

The tests performed can be of different forms. Most of the time, unit tests are written to make sure that small building blocks (processing units) are tested efficiently and are producing the expected output. Once the unit tests are all passing, more global and targeted tests can be performed to make sure the overall system is performing as expected. In the case of unit testing, tests can be as simple as comparing returned values from expected values. For instance if a given processing unit or filter is taking an array of numbers and computing its mean, the expected value is a known values given a known input and can be made part of the validation process. However, testing can get a lot more complicated when the range of input values varies greatly. For instance, to test an I/O method that reads any DICOM images, the validation scheme must try all the possible DICOM formats and check that the reading and writing is performed correctly. As one can see, this would require a significant amount of time and processing power. Now, one can think about testing a registration algorithm that takes two images as input. In that case, the validation should be designed to run a subset of the input space otherwise the possible combinations are endless.

We have seen how inputs of a unit test can affect the validation of an algorithm, and we should notice that this is also true for the output of a test. For instance, given a filter that blurs an input image and produces an output image, the best way to validate this algorithm would be to compare the resulting image with an expected image. Let's now imagine a segmentation algorithm which uses random seed points to initialize its processing. In that case, because the initialization is random, the resulting segmentation given a fix input might be different from run to run, and therefore to completely validate the algorithm more than one expected output should be compared.

As we have seen algorithm validation can be challenging. Even the selection of the right input and outputs can remain difficult; fortunately several validation tools, commercial and open-source exist to help with the testing development. Among the commercial tools, Rational and Microsoft have been developing widely used products. On the other hand, CMake[2] and CDash[3] have been used for several open-source project. These systems integrate a source code management tool with a software validation platform. The source code is compiled on several platforms on a regular basis (at least every night) and the result of the validation is submitted to the "dashboard" which provides a unified view across the clients. Developers then check the dashboard and fix the code as required.

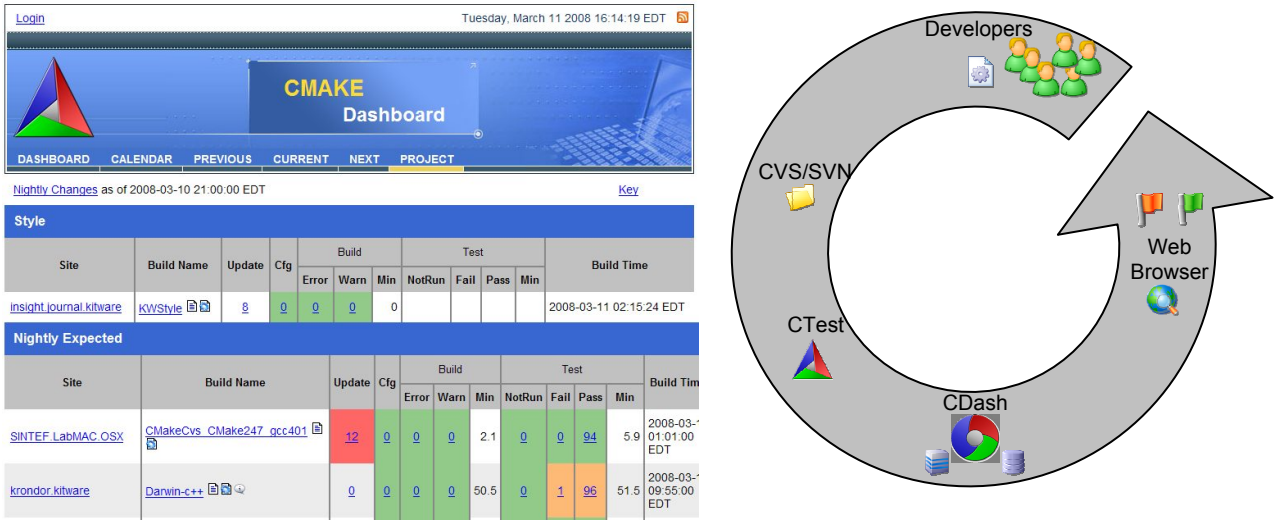

Fig. 1. Left: Main page of the Dashboard provided by CDash. Right: Development cycle in an extreme programming environment.

2.2 Tracking devices

As we have seen in the previous section, unit testing is directly affected by the nature of the inputs. Tracking devices are difficult to tests because of their wide range of variations. Let's take a simple example of a program that prints out the latitude and longitude from a USB global positioning (GPS) device. In order to validate the program, one can think about using a simulation file which for instance will simulate the USB protocol and data and would test the program. One can notice that using simulation file is useful because the range of data can be expanded easily and they are usually easy to create. Moreover, simulation files also do not require the tracking system to be plugged in when running the test. However simulation files have some limitations. First, they cannot replace manual interaction because the precision of the tracking device cannot be easily embedded in a simulation file. Second, to test the system in a real scenario, simulation files are just too limited. For instance, imagine the same GPS system while driving in a tunnel and the signal is lost. Real scenarios will always provide an optimal testing environment. And real world tracking avoid this disconnection between the tracking device and the software to be tested. Finally, simulation files often have to be generated using the original tracking devices and manually updating, or altering, a simulations file is often difficult because a low level protocol is often written on file. This limitation means that in order to test a new feature of the tracking device, a new simulation file should be created and this cannot be done programmatically.

3. CONCEPTION AND REALISATION

We have created an open-source framework for testing tracking devices in a real environment. Our framework is limited to small range systems but allows continuous testing in a programmatic manner.

Next we demonstrate how we use the Lego Mindstorms [4] as a basic technology for testing.

3.1 Lego Mindstorms

The Lego Mindstorms set has been used for teaching and experimenting with robotic due to its simplicity of use and its affordable price. The set can be bought for less than 250USD. Lego Mindstorms is a line of Lego sets combining programmable units such as electric motors and sensors. The hardware and software of the Lego Mindstorms have been originated by the MIT media lab and have been released to the public for the first time in 1998. In 2006, Lego released a new version of the system named NXT. The NXT set, that we use, is composed of three servo motors and four sensors for touch, light, sound and distance. More sensors can be bought individual such as color, compass, infrared link,

ultrasonic, etc… The NXT motors and sensors have been known to be reliable and of good enough resolution for robotic applications.

The success of the Lego Mindstorm™ has led to several application programming interfaces (API) to communicate with NXT. Lego provides an official NXT API in Basic, C++ and ASM but a lot of third-party interfaces exist and provide support for other programming languages such as C#, Java, Python, Lua, Perl and much more.

Fig. 2. Left: Example of our robot arm. Right: Lego Mindstorms set with his 3 motors and 4 sensors.

3.2 Testing framework

We have build a testing framework inspired by Danielle Pace's Image-Guided Therapy Robot[5]. We have taken the base Lego infrastructure and adapted it for the testing of tracking devices. Our simple robot offers 3 degrees of freedom and covers a 12 inches cube range. Our robot consists of an arm which supports the tracking device. By programming the two servo motors, the arm can be moved to any point in the 3D space. One can notice that the robot can be extended to support more degrees of freedom and cover more space.

The main issue we encountered with the servo motor comes from the precision of the movement which do not allow for highly reproducible paths. For instance, if one sends a command to the motor to move 10 degrees clockwise and then to move back 10 degrees counter clockwise, the final position is more likely to be off by at least one degree. Fortunately several options exist. First, the testing framework can avoid using relative position of the arm and reinitialize itself every time it starts. However this solution does not prevent possible drift in the long run. Second, this overshooting effect is amplified when the motor stops at high velocity. Fortunately, the speed of the servo motor can be controlled from the USB device and, to prevent this overshoot, the motor should start at high velocity and reduce its speed before stopping. Finally, the arm can self-calibrate itself before each run, ensuring that the main initial position is known and well defined. We use a combination of all the techniques cited above in our framework. First we recommend to programmers to always use relative coordinates from the tracking devices. This makes the testing framework independent from the starting position. Next we specifically implemented a way to rotate the motor such that the motor travels most of the distance at high velocity and decrease its speed when approaching its final position. Third we have added an auto-calibration system for the arm. The calibration system uses the light and touch sensors. First we rotate the arm in the horizontal (X-Y) plane and stop when the arm is in front of the light sensor. Next we move the arm in the vertical (Y-Z) plane until the arm push the touch sensor. This provides a good enough calibration of the system and once the arm is calibrated developers can use the API to move the arm and start the testing.

3.3 Application Programming Interface

Our API is written in C++ and is based on the open-source NXT++ library[6] and uses libUSB[7] for the USB interface. The API is cross-platform (linux and windows) and is available from the NaMic Sandbox[8]. The API provides a simple low-level interface for the Lego Mindstorm NXT. Combined with the robotic arm described previously, the API allows developers to move the arm in any direction and record the expected position and orientation of the tracking device.

The API first establishes a connection with the NXT using the USB driver. Once the connection is established, the developer can move the arm at different locations and also control the different sensors. The movement of the robotic arm is controlled by two servo-motor which returns the current orientation since motors can be used as rotational sensors as well. Therefore, to move the arm, one should give the degree increment as well as the speed of rotation (on a scale from 1 to 10). The API is using an object-oriented programming concept and is very simple to use.

Our API overcomes the limitation of the simulation files by providing a programmable framework for testing tracking devices. Thus, a software developer in Europe can check in a new test pattern for a tracking device without having the real tracking device close by. Once compiled, the test can be run and the Lego arm can start moving and testing the application, even at a different location around the globe. Moreover, the main advantage of the API is that it can be integrated into any unit tests or application testing environment. For instance, in order to test that the movement of the tracker is correctly linked to the movement of a sphere on the screen, a test can create the sphere, link the sphere to the tracker, ask the arm to move to a specific position and finally check if the resulting expected position of the sphere is correct.

```
#include "NXT_USB.h"

int main( int argc, char * argv [] )
{
  NXT_USB nxtUSB;

  // Establish communicationg if NXT
  if(!nxtUSB.OpenLegoUSB())
    {
    return 0;
    }

  // Set the light sensor
  unsigned int lightSensorPort = 1;
  nxtUSB.SetSensorLight( lightSensorPort, true );
  // Get the light sensor level
  const int lightLevel = nxtUSB.GetLightSensor(lightSensorPort-1);

  // Move the vertical motor up
  nxtUSB.MoveMotor(0,30,10);
  // Move the horizontal motor to the left
  nxtUSB.MoveMotor( 1,  -30, 10 );

  return 1
}
```

Fig. 3. Sample C++ code showing the simplicity of use of the NXT library.

4. APPLICATION TO IMAGE-GUIDED SURGERY

We are currently using our system for the development of the Image-Guided Surgery Toolkit (IGSTK)[9]. We describe this toolkit next.

4.1 The Image-Guided Surgery Toolkit

IGSTK is a high-level component-based framework providing common functionality for image-guided applications. IGSTK has been funded by NIBIB and NIH and development has been led by the ISIS center at Georgetown University and Kitware Inc. Several other groups are contributing to the development of the toolkit including Arizona State University and SINTEF in Norway. IGSTK is distributed as open-source software under a BSD license, it allows unrestricted use.

IGSTK provides the ability to (a) read and display medical images in DICOM form, (b) track instruments from magnetic or optical trackers, (c) visualize in real-time the current simulation and (d) apply image processing algorithms. IGSTK is written in C++ and the toolkit is built on top of open-source toolkits, ITK, VTK and FTLK. Moreover, IGSTK supports a large range of trackers: Micron, Aurora, Polaris and Flock of Birds and provides a unified interface to communicate with them. The cornerstone of IGSTK is robustness and in this context, our new framework has been improving the overall testing of the toolkit.

4.2 Nightly testing

IGSTK follows a software process based on the extreme programming paradigm. The main development is done in the "Sandbox" where new features and improvements added on a daily basis. Once the Sandbox is stable and the feature requirements are validated, the code is moved to the main repository. As part of the testing of the sandbox and the main repository, we have been using simulations files as described previously. Since IGSTK developers, as for most open-source software, are located in cities all over the world, it was difficult to generate a consistent testing framework, especially for tracking devices. The main issue we encounter was the generation of the simulation files which would have to be created every time a modification is done to the application. This procedure was not tractable. However now, using our framework, developers of applications can quickly implement a test and program the Lego arm to move at a certain position and make sure the algorithm and/or visualization produces the right result.

Furthermore, every night, some of the 100+ unit tests ran by CTest are using the Lego framework and move the robotic arm, comparing the resulting position with the expected value and reporting any failures to the dashboard.

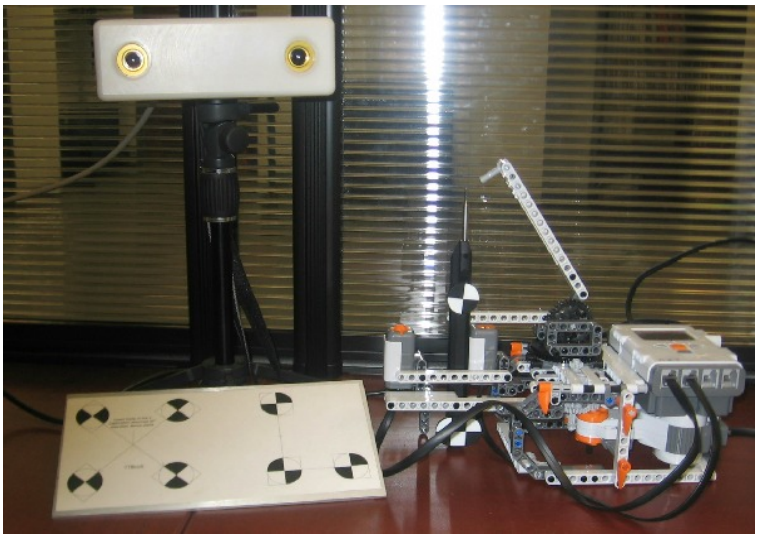

Fig. 4. Our Lego robot operational for the testing of the Image-Guided Surgery Toolkit with the Micron Tracker.

5. CONCLUSION

We have shown that testing software that use tracking devices can be easily performed using a low-cost, open-source framework with the help of a Lego Mindstorms set. We have also demonstrated that the current framework can be applied to the testing of the Image-Guided Surgery Toolkit in the context of the extreme programming software process. The framework remains extensible and can be used for any applications which need medium range physical testing.

We are also conscious that our framework has some limitations. For instance, the spatial range is not as important as one could hope. This is due to the limitation of the Lego Mindstorms set that cannot support a longer arm. Also, the testing system requires the tracking device to be running at all times. One option would be to use a USB powered switch which would start the tracker programmatically. Moreover, the current system currently does not support multiple devices, which would be useful to track multiple tools. We are still improving the current framework and we hope to help programmers develop more robust software, especially those using tracking devices.

This work was funded by NIBIB/NIH grant R01 EB007195. The content of this manuscript does not necessarily reflect the position or policy of the U.S. Government. All of the software is freely available for download and can be used in research or commercial applications. More information can be found on the website at http://www.igstk.org.

REFERENCES

[1] Schroeder, Ibanez, Martin. Software Process: "The Key to Developing Robust, Reusable and Maintainable Open-Source Software". IEEE International Symposium on Biomedical Imaging: Macro to Nano, (2004).

[2] Martin K. and Hoffman B., "Mastering CMake: A Cross-Platform Build System", Kitware Inc., (2003) http://www.cmake.org

[3] CDash: an open-source, web-based testing server: http://www.cdash.org

[4] Lego Mindstorms: http://wiki.na-mic.org/Wiki/index.php/LEGO_IGT_and_Medical_Robotics_Tutorial

[5] Pace D.F., Kikinis R., Hata N. "An accessible, hands-on tutorial system for image-guided therapy and medical robotics using a robot and open-source software", The Insight Journal. Jan 2008. http://hdl.handle.net/1926/567

[6] The NTXPP Library: http://nxtpp.sourceforge.net

[7] The LibUSB Library http://libusb.sourceforge.net

[8] Accessing the NaMic Sandbox: http://wiki.na-mic.org/Wiki/index.php/Engineering:SandBox

[9] Gary, K.; Ibanez, L.; Aylward, S.; Gobbi, D.; Blake, M.B.; Cleary, K. "IGSTK: an open source software toolkit for image-guided surgery", IEEE Computer Volume 39, Issue 4, April 2006 Page(s): 46 – 53.