Extending the tracking device support in the Image-Guided Surgery Toolkit (IGSTK): CamBar B2, EasyTrack 500, and Active Polaris

Release 0.00

Özgür Güler[1], Zoltan R. Bardosi[1], Murat Ertugrul[2], and Wolfgang Freysinger[1]

May 9, 2011
[1]4D Visualization Laboratory
Univ. ENT Clinic
Innsbruck Medical University
Anichstraße 35,
6020 Innsbruck
[2]Vienna University of Technology
Faculty of Informatics
Erzherzog-Johann-Platz 1/E180
1040 Vienna

Abstract

The Image-Guided Surgery Toolkit (IGSTK) provides tracker interfaces for various tracking devices. The tracker component of IGSTK was extended with three new tracking interfaces: CamBar B2, EasyTrack 500, and Active Polaris. Using an IGSTK application we evaluated the precision of each of the tracking systems. Based on our evaluation we conclude that all tracking systems are sufficiently accurate for ENT procedures.

Contents

1 Introduction

Position measurement systems are one of the key technologies in Image-Guided Systems (IGS). They enable the localization of surgical instruments and anatomical structures in space. According to requirements, different measurement technologies are available, however optical and electromagnetic tracking systems are the most established [1;2]. Optical tracking systems often have better accuracy whereas electromagnetic systems do not impose the “line-of-sight” requirement. The Image-Guided Surgery Toolkit (IGSTK) provides tracking system APIs for both types of measurement systems [3].

1.1 Tracker Component in IGSTK

IGSTK abstracts a position measurement device using a class for the tracker and another for the tracker tool. The igstk::Tracker class is an abstract representation of a tracking device. It is responsible for establishing the connection, initializing the device, starting and stopping tracking. To specify a concrete device the pure virtual member functions of the igstk::Tracker class beginning with “internal” have to be implemented (e.g. “InternalOpen()” for initializing the tracker). This is where the device specific behavior is implemented. The igstk::Tracker class can handle several tracker tools abstracted by the igstk::TrackerTool class. The abstract tracker tools are held in a container inside the igstk::Tracker class. Figure 1 shows the tracker component and its interaction with other IGSTK components.

According to the frequency, set by the user, a pulse generator triggers updates. The igstk::Tracker class copies the current transformations from an internal buffer to the appropriate TrackerTool object. Since the SpatialObjects which represent the TrackerTool in the scene are connected over the coordinate system mechanism in IGSTK with the TrackerTool, they follow the change in position too. Operations such as triggering an update from the pulse generator, passing new transforms to the tracker tools and changing the position of the spatial objects happen inside the application’s main thread. The communication with the concrete device and retrieving the current position of the rigid body is performed by an additional

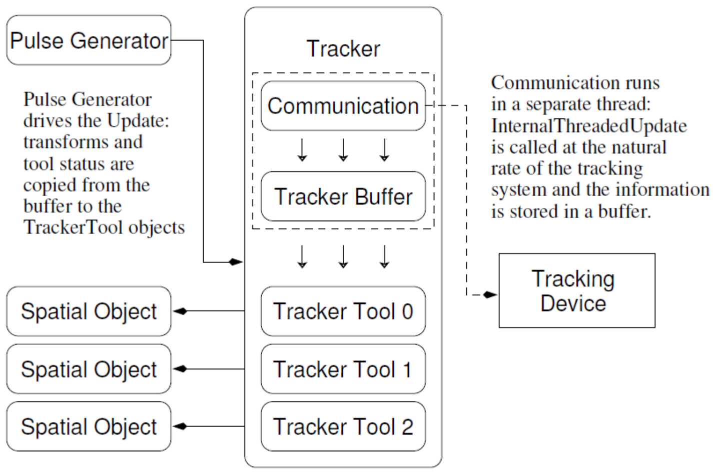
Pulse Generator
Pulse Generator drives the Update: transforms and tool status are copied from the buffer to the TrackerTool objects
Tracker
Communication
Tracker Buffer
Communication runs in a separate thread: InternalThreadedUpdate is called at the natural rate of the tracking system and the information is stored in a buffer.
Tracking Device
Spatial Object
Spatial Object
Spatial Object
Tracker Tool 0
Tracker Tool 1
Tracker Tool 2

Figure 1: IGSTK tracker component architecture.

thread. The components participating in this separate thread are shown in Figure 1 inside the dashed line.

Currently IGSTK supports the following tracking devices: Aurora and Polaris from Northern Digital Inc. (Waterloo, ON, Canada) [4], flock of birds and 3D Guidance from Ascension Technology Corp. (Milton, VT, USA) [5], InfiniTrack from Atracsys LLC. (Le Mont-sur-Lausanne, Switzerland)[6] and Micron Tracker from Claron Technology Inc. (Toronto, ON, Canada)[7].

IGSTK also provides tracker classes for test and debug purposes such the MouseTracker, QMouseTracker and SimulatedTracker.

In this work we add support for the following tracking systems: CamBar B2 from AXIOS 3D Services GmbH (Oldenburg, Germany), EasyTrack 500 from Atracsys LLC., (Le Mont-sur-Lausanne, Switzerland)[6], and the original Polaris system, "Polaris Classic", from Northern Digital Inc. (Waterloo, ON, Canada)[4].

The CamBar B2 and EasyTrack 500 were previously not supported in IGSTK. The Polaris Spectra and Vicra are supported in IGSTK, but the "Polaris Classic", is not. This is due to changes in the system's API. The "Polaris Classic" is an active optical tracking system that utilizes wired rigid bodies. The tracked rigid bodies are connected to one of three physical ports on the Tool Interface Unit. The three ports are accessed using the appropriate port number. The NDI command interpreter currently implemented in IGSTK only supports the updated API which uses the concept of handles. This API is shared by the Polaris and Aurora systems and allows multiple tools to be connected to the same physical port (i.e. two 5 degree of freedom electromagnetic sensors on one physical port). Therefore addressing the rigid bodies using the port number is not possible and a handle concept is used instead. Figure 2 shows the updated tracker hierarchy in IGSTK.

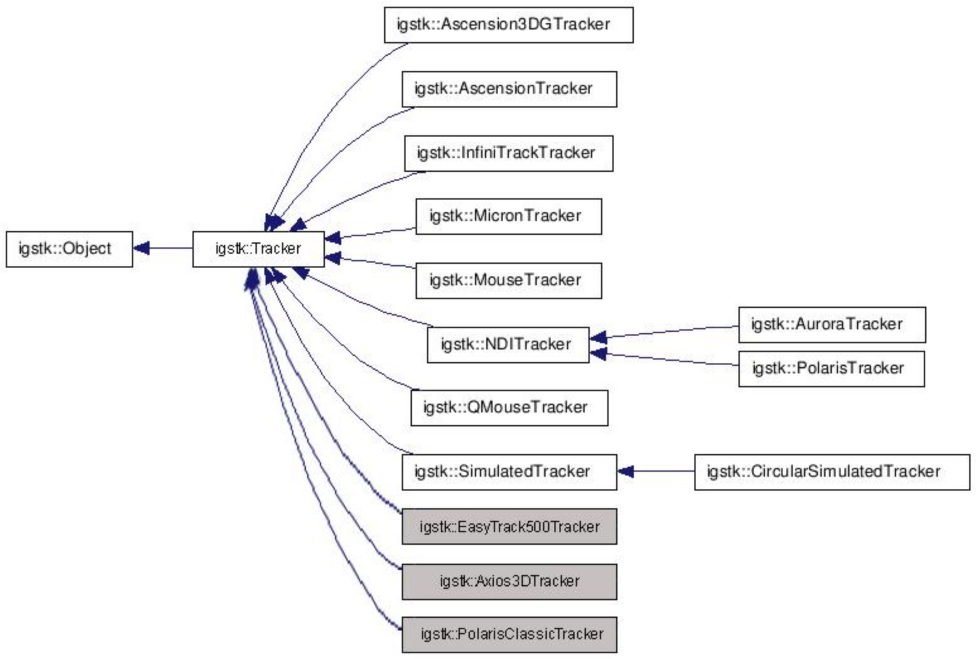

Figure 2: Inheritance diagram for the igstk::Tracker classes, additions highlighted in grey.

2 Materials

Three new tracker interfaces for the optical tracking systems listed below were integrated into IGSTK and evaluated:

- CamBar B2, AXIOS 3D Services GmbH, Germany (http:www.axios3d.de)
- EasyTrack 500, Atracsys LLC., Switzerland (http:www.atracsys.com)
- "Polaris Classic", Northern Digital Inc., Ontario, Canada (http:www.ndigital.com)

The CamBar B2 is an optical tracking device designed by Axios 3D that supports passive tracking of retro-reflective spherical markers, flat markers, bore holes, inkjet printed markers, and laser engravings. Communication with the device is done by network cable (RJ-45). The workspace is frustum shaped and begins at 800 mm and ends at 2500 mm from the tracking systems origin along the viewing direction. The system can track up to eight tools simultaneously. New reference frames can be readily created using the retro-reflective spherical markers or inkjet markers. Figure 3 shows this system.

The EasyTrack 500 device is an optical measurement system designed by Atracsys that supports active tracking of rigid-bodies with LEDs. Communication with the device is done via a USB connection. The workspace is frustum shaped and begins at 220 mm and ends at 1000 mm from the tracking systems origin along the viewing direction. It supports up to four internal (plugged-in) and eight external tools. A kit for building custom markers is also available. The active tracker tools are automatically recognized by the device, thus they need no additional configuration. It is shipped with open-source drivers for Windows 7, Windows XP, GNU/Linux and Mac OS X. The system uses three one dimensional cameras to estimate the tool positions. Figure 4 shows this system.

The “Polaris Classic” is an optical tracking device designed by NDI that supports active tracking of LEDs mounted on rigid-bodies. Communication with the device is done by serial interface (RS-232). The workspace is silo shaped and begins at 1400 mm and ends at 2500 mm from the tracking systems origin along the viewing direction. The system can track up to three tools simultaneously. Figure 5 shows this system.

Table 1 summarizes the technical specifications of these systems.

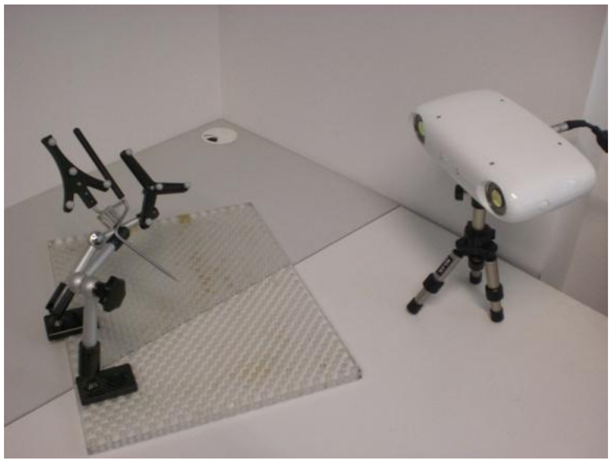

Figure 3: CamBar B2, AXIOS 3D, Setup shows tracking camera on the right, dynamic reference frame on the left and the probe on the outer left.

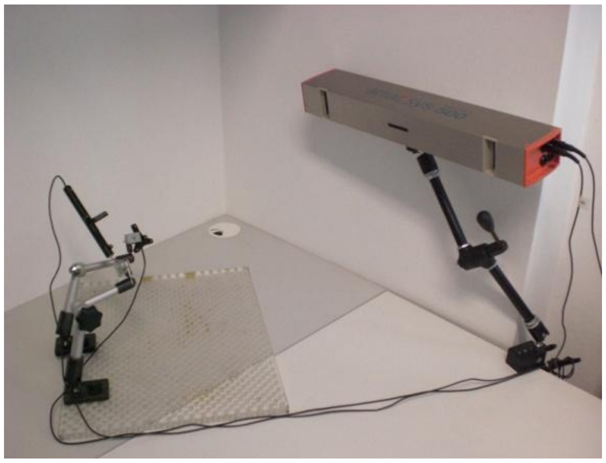

Figure 4: EasyTrack 500, Atracsys

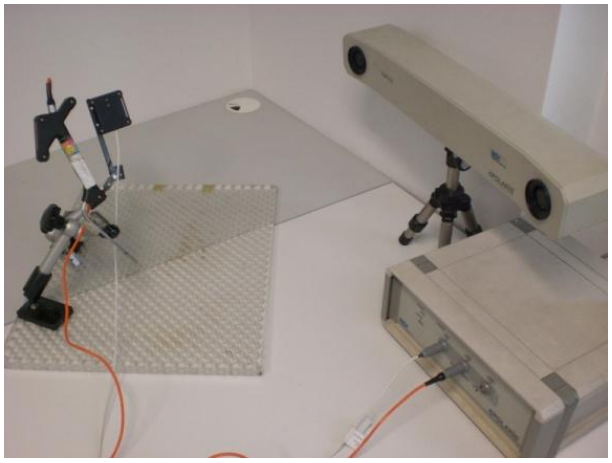

Figure 5: Polaris Classic, Northern Digital Inc.

Model Name	CamBar B2 Typ 1.2	EasyTrack 500	Polaris CLASSIC
Manufacturer	AXIOS 3D Services GmbH, Oldenburg, Germany	Atracsys LLC., Switzerland	NDI, Ontario, Canada
Working Volume (min/ max distance to the camera)	800 mm / 2500 mm* *Built to specification	220 mm / 1000 mm	1400 mm / 1900 mm
Accuracy inside Working Volume (manufacturer data sheet)	50 µm	0.2 mm	RMS 0.35 mm
Dimensions in mm (W/H/D):	285 / 75 / 163	554 / 98 / 65	585 / 76 / 115
Weight:	2.2 kg	1.6 kg	2 kg
Frequency (max)	64 images/sec	300 LEDs/sec	60 Hz
Interface	Network	USB 1.1 or 2.0	RS-232
Platform	Win32	Win32, UNIX, OS X	Win32, UNIX, OS X
Approvals	CE certified medical product	For research only	For research only

Table 1 Detailed manufacturer information about each tracking system.

We next evaluate the accuracy of the tracking devices.

3 Experimental Evaluation

Using an IGSTK application we evaluated the precision of each of the tracking systems. Figures 3, 4 and 5 show the setup for the experiments. A dynamic reference frame (DRF), and a probe, were fixed on a table. We performed three data acquisitions. First we acquired 1000 measurements from a single marker (one LED of a "Polaris Classic" or Easy Track 500 rigid-body, one retro-reflective spherical marker of a CamBar B2 rigid-body) using custom non IGSTK programs. Then we acquired 1000 pose measurements of a DRF. Finally, we acquired 1000 pose measurements for both tools, DRF and probe. Data was acquired after system warm-up. The measured markers or tools were positioned in the center of the working volume. Tracking after system warm-up and at the center of the working volume should ensure stable and reproducible measurements.

The EasyTrack 500 and CamBar B2 had comparable results in all experiments, including single marker, single DRF and probe-to-DRF measurements. The "Polaris Classic" was the least stable/accurate. Table 2 shows the standard deviations of all experiments in the X, Y and Z directions. Table 3 shows the maximum error in the X, Y and Z directions for single marker, DRF, and probe to DRF measurements, respectively. Also here the EasyTrack 500 and the CamBar B2 had comparable results in all experiments.

Standard Deviation		**Axios 3D B2**			**EasyTrack 500**			**Polaris CLASSIC**	
[mm]	**X**	**Y**	**Z**	**X**	**Y**	**Z**	**X**	**Y**	**Z**
Single Marker	0.0068	0.0057	0.0250	0.0052	0.0061	0.0135	0.0274	0.0086	0.0430
DRF	0.0123	0.0119	0.0212	0.0040	0.0043	0.0156	0.0166	0.0148	0.0283
DRF to Probe	0.0132	0.0278	0.0890	0.0189	0.0324	0.0208	0.2971	0.0868	0.1581

Table 2: Precision of the three optical tracking systems given in the X, Y and Z directions

Max Error		**Axios 3D B2**			**EasyTrack 500**			**Polaris P1**	
[mm]	**X**	**Y**	**Z**	**X**	**Y**	**Z**	**X**	**Y**	**Z**
Single Marker	0.0172	0.0125	0.0849	0.0190	0.0250	0.0418	0.0573	0.0695	0.1272
DRF	0.0296	0.0293	0.0633	0.0200	0.0137	0.0945	0.0661	0.0795	0.0869
DRF to Probe	0.0477	0.0896	0.3152	0.0460	0.2398	0.1465	0.6308	0.1439	0.2512

Table 3: Maximum error of the three optical tracking systems given in the X, Y and Z directions

4 Sample Code

An IGSTK application was used to log the tracker measurements using either one or two connected tracker tools into a text file. The application needs to initialize the tracker, and attach two tracker tool objects, so that we can measure the position and orientation of a tool w.r.t. the tracker, or another reference tool. As a first step, we define an abstract 'TrackerInitializer' class, to define a device independent initialization scheme.

Later we will specialize this class to implement the device dependent initializer classes.

This is the definition of the class 'TrackerInitializer':

```
#include <list>
#include <string>

#include "igstkTrackerTool.h"

class TrackerInitializer {
public:
      TrackerInitializer()
      {
      }

      ~TrackerInitializer();

      virtual igstk::Tracker::Pointer CreateTracker() = 0;
      virtual igstk::TrackerTool::Pointer CreateTrackerTool( int id ) = 0;

      static std::list<std::string> GetAvailableTrackerTypes() {
            m_trackerTypes.clear();
             register the available tracker types
#ifdef IGDT_USE_EASYTRACK
            m_trackerTypes.push_back( "EasyTrack500" );
#endif
#ifdef IGDT_USE_CLASSICPOLARIS
            m_trackerTypes.push_back( "ClassicPolaris" );
#endif
#ifdef IGDT_USE_AXIOS3D
            m_trackerTypes.push_back( "Axios3D" );
#endif
            return m_trackerTypes;
      }

      static TrackerInitializer *CreateTrackerInitializer( std::string type
);
protected:
      static std::list<std::string> m_trackerTypes;
};
```

To use the Atracsys EasyTrack 500 device, we create a specialization of the TrackerInitializer class, using the EasyTrack 500 IGSTK implementation. In the Logger application, we used several CMake macro definitons to turn the specific tracker implementations on and off.

```
#ifdef IGDT_USE_EASYTRACK
#include "igstkAtracsysEasyTrack.h"
#include "igstkAtracsysEasyTrackTool.h"

class EasyTrackInitializer : public TrackerInitializer
{
public:
      EasyTrackInitializer() : TrackerInitializer() {}
      ~EasyTrackInitializer() {}

      igstk::Tracker::Pointer CreateTracker()
      {
```

EasyTrack 500 instance is created here.

```
            return igstk::AtracsysEasyTrack::New().GetPointer();
      }
```

We define two tools and instantiate appropriate TrackerTool objects.

```
      igstk::TrackerTool::Pointer CreateTrackerTool( int id )
      {
            switch( id ){
                  case 0:
                  {
                        igstk::AtracsysEasyTrackTool::Pointer tool =
                        igstk::AtracsysEasyTrackTool::New();
                        tool->RequestSetPortNumber(0);
                        return tool.GetPointer();
                  }
                  case 1:
                  {
                        igstk::AtracsysEasyTrackTool::Pointer tool =
                        igstk::AtracsysEasyTrackTool::New();
                        tool->RequestSetPortNumber(1);
                        return tool.GetPointer();
                  }
            }

            return NULL;
      }
};

#endif
```

To use the “Polaris Classic” tracker, we create a specialization of the TrackerInitializer class, using the “Polaris Classic” IGSTK implementation.

```
#ifdef IGDT_USE_CLASSICPOLARIS
#include "igstkSerialCommunication.h"
#include "igstkPolarisClassicTracker.h"
#include "igstkPolarisTrackerTool.h"

class ClassicPolarisInitializer : public TrackerInitializer
{
      ClassicPolarisInitializer() : TrackerInitializer() {}
      ~ClassicPolarisInitializer() {}

      igstk::Tracker *CreateTracker()
      {
```

After creating the SerialCommunication object the “Polaris Classic” instance is created.

```
            // create the communication object
            igstk::SerialCommunication::Pointer  serialComm =
            igstk::SerialCommunication::New();

            serialComm->SetPortNumber( igstk::SerialCommunication::PortNumber3 );
            serialComm->SetParity( igstk::SerialCommunication::NoParity );
            serialComm->SetBaudRate( igstk::SerialCommunication::BaudRate115200 );
            serialComm->SetDataBits( igstk::SerialCommunication::DataBits8 );
            serialComm->SetStopBits( igstk::SerialCommunication::StopBits1 );
            serialComm->SetHardwareHandshake(
                   igstk::SerialCommunication::HandshakeOff );
            serialComm->SetTimeoutPeriod(100);
            serialComm->SetCaptureFileName(
            "RecordedStreamBySerialCommunicationTest.txt" );
            serialComm->SetCapture( true );
            if( serialComm->GetCapture() != true )
            {
                   std::cout << "Set/GetCapture() failed" << std::endl;
                   std::cout << "[FAILED]" << std::endl;
            }

            std::cout << "CaptureFileName: "
                          << serialComm->GetCaptureFileName() << std::endl;

            serialComm->OpenCommunication();

            igstk::PolarisTracker::Pointer tracker = igstk::PolarisTracker::New();
            tracker->SetCommunication(serialComm);

            return tracker.GetPointer();

      }
```

We define two tools and instantiate appropriate TrackerTool objects.

```
      igstk::TrackerTool* CreateTrackerTool( int id )
      {
                  switch( id ){
                  case 0:
                  {
                        igstk::PolarisTrackerTool::Pointer tool =
                        igstk::PolarisTrackerTool::New();
                        tool->RequestSelectWiredTrackerTool();
                        tool->RequestSetPortNumber(0);
                        return tool.GetPointer();
                  }
                  case 1:
                  {
                        igstk::PolarisTrackerTool::Pointer tool =
                        igstk::PolarisTrackerTool::New();
                        tool->RequestSelectWiredTrackerTool();
                        tool->RequestSetPortNumber(1);
                        return tool.GetPointer();
                  }
            }
            return NULL;

      }
};

#endif
```

Specialization for the Axios3D tracker:

```
#ifdef IGDT_USE_AXIOS3D
class Axios3DInitializer : public TrackerInitializer
{
      Axios3DInitializer() : TrackerInitializer() {}
      ~Axios3DInitializer() {}

      igstk::Tracker *CreateTracker()
      {
```

We instantiate the Axios3D tracker and set the appropriate configuration files of each rigid-body.

```
            igstk::Axios3DTracker::Pointer tracker = igstk::Axios3DTracker::New();
            tracker->RequestLoadLocatorsXML("C:/workspace/IGDTTrackerLogger/DRF.xml");
            tracker->RequestLoadLocatorsXML(
            "C:/workspace/IGDTTrackerLogger/Probe.xml");
            tracker->setVirtualMode(false);
```

```
		return tracker.GetPointer();

	}
```

We define two tools and instantiate appropriate TrackerTool objects.

```
	igstk::TrackerTool* CreateTrackerTool( int id )
	{
		switch( id ){
		case 0:
		{
			igstk::Axios3DTrackerTool::Pointer tool =
			igstk::Axios3DTrackerTool::New();
			tool->RequestSetMarkerName("PROBE");
			return tool.GetPointer();
		}
		case 1:
		{
			igstk::Axios3DTrackerTool::Pointer tool =
			igstk::Axios3DTrackerTool::New();
			tool->RequestSetMarkerName("DRF");
			return tool.GetPointer();
		}
		return NULL;

	}
};

#endif
```

We use the Factory design pattern to allow the creation of the device specific TrackerInitializer classes using a string as the tracker identifier.

```
TrackerInitializer *TrackerInitializer::CreateTrackerInitializer( std::string
type )
{
#ifdef IGDT_USE_EASYTRACK
	if( type == "EasyTrack500" ){
		return new EasyTrackInitializer();
	}
#endif
#ifdef IGDT_USE_CLASSICPOLARIS
	if( type == "ClassicPolaris" ){
		return new ClassicPolarisInitializer();
	}
#endif
#ifdef IGDT_USE_AXIOS3D
	if( type == "Axios3D" ){
		return new Axios3DInitializer();
```

```
      }
#endif

      return NULL;
}
```

We next look at the implementation details of the TrackerLogger application. One important concept is that we hold all the available specific trackers as a list of their identifiers. We define a static method in the TrackerInitializer class so that we can query the available tracker implementations. It also requires a static member m_trackerTypes, which has to be defined here.

```
#include <iostream>
#include <fstream>

#include "igstkTracker.h"
#include "igstkTrackerTool.h"
#include "igstkTransform.h"
#include "igstkTransformObserver.h"
#include "igstkAxesObject.h"

#include "IGDTTrackerLogger.h"

// data storage of available trackers
std::list<std::string> TrackerInitializer::m_trackerTypes;
```

The main function of our application defines the application logic in a tracker independent way using only the TrackerInitializer interface to create and initialize the appropriate IGSTK Tracker object along with the required TrackerTool based objects. In the followings we show how to query the identifiers of the available tracker implementations, and how to initialize a tracker with the given identifier.

First, we query the identifier list of the available tracker implementations.

```
std::list<std::string> trackerIDs =
                  TrackerInitializer::GetAvailableTrackerTypes();
for( std::list<std::string>::iterator it = trackerIDs.begin();
                                          it != trackerIDs.end(); ++it )
{
                  std::cout << "\t" << *it << std::end
}
```

Once we have the identifier for the desired tracker, we can create the device specific TrackerInitializer derived object using the static factory method from the TrackerInitializer class.

```
TrackerInitializer *trackerInitializer =
                  TrackerInitializer::CreateTrackerInitializer(trackerID);
```

Now we have the specialization of the initializer for the desired tracker object, so we can request the objects required by the IGSTK framework.

```
igstk::Tracker::Pointer tracker = trackerInitializer->CreateTracker();
tracker->RequestSetTransformAndParent( transform, worldReference );

tracker->RequestOpen();

igstk::TrackerTool::Pointer tool0 =
                              trackerInitializer->CreateTrackerTool(0);
tool0->RequestConfigure();
tool0->RequestAttachToTracker( tracker );

igstk::TrackerTool::Pointer tool1;

if( bRefMode ){
      tool1 = trackerInitializer->CreateTrackerTool(1);
      tool1->RequestConfigure();
      tool1->RequestAttachToTracker( tracker );

      tracker->RequestSetReferenceTool(tool1);
}
```

Everything is in place, we can use the standard IGSTK tracker framework with the objects we received from the tracker initializer.

```
std::cout << "start tracking\n"; std::cout.flush();
tracker->RequestStartTracking();
std::cout << "start tracking done\n";     std::cout.flush();
```

5 Discussion & Conclusion

IGSTK enables the rapid development of image-guided surgery applications. A key component of these is the tracking device. IGSTK already provides tracker interfaces for various tracking devices. We have extended the tracker component of IGSTK with three new tracking interfaces. The EasyTrack 500 and CamBar B2 systems come with C++ libraries that facilitate data acquisition while the Polaris system provides a more primitive string based communication protocol. However the CamBar B2 has drivers for Windows XP/Vista/7 only. Having the APIs the implementation of IGSTK classes was straight forward. This implementation enables easy data acquisition.

Based on our evaluation we conclude that all tracking systems are sufficiently accurate for ENT procedures where the required accuracy is approximately 1-2 mm (RMS). It should be noted that the Polaris system evaluated in this study has been in use since 1996. Empirical system accuracy is still stable.

The implementations described in this manuscript can be downloaded from the IGSTK sandbox, and are released under the toolkit’s BSD license.

The IGSTK sandbox can be checked out directly from CVS as follows:

```
cvs -d :pserver:anonymous@public.kitware.com:/cvsroot/IGSTK login
```

password: `igstk`

```
cvs -d :pserver:anonymous@public.kitware.com:/cvsroot/IGSTK co IGSTKSandbox
```

ACKNOWLEDGMENT

We would like to thank Dr. Ziv Yaniv for his help editing this manuscript.

This work was supported by the Austrian Science Foundation under Grant P 20604-B13 (Özgür Güler) and the Austrian National Bank, Jubilee Fond, Project no. 13003 (Zoltan Bardosi).

Reference

[1] Z. Yaniv and K. Cleary, "Image-Guided Procedures: A Review," Imaging Science and Information Systems Center, Georgetown University, Washington, DC,CAIMR TR-2006-3, 2006.

[2] K. Cleary and T. M. Peters, "Image-guided interventions: technology review and clinical applications," *Annu. Rev. Biomed. Eng*, vol. 12, pp. 119-142, Aug.2010.

[3] A. Enquobahrie, D. Gobbi, M. W. Turek, P. Cheng, Z. Yaniv, F. Lindseth, and K. Cleary, "Designing tracking software for image-guided surgery applications: IGSTK experience," *Int. J. Computer Aided Radiology and Surgery*, vol. 3, pp. 395-403, 2008.

[4] Northern Digital Inc. (Waterloo, ON, Canada), http://www.ndigital.com/medical , Date Accessed: 16-6-2011

[5] Ascension Technology Corp. (Milton,VT,USA), http://www.ascension-tech.com , Date Accessed: 16-6-2011

[6] Atracsys LLC. (Le Mont-sur-Lausanne, Switzerland), http://www.axios3d.de/index_en.html , Date Accessed: 16-6-2011

[7] Claron Technology Inc. (Toronto, ON, Canada), http://www.clarontech.com , Date Accessed: 16-6-2011