MEDTRONIC

POLARIS SIMULATOR

ZACHERY TILLOTSON + NATHAN MCNEEL + NICHOLAS GEANETTA

SUMMER FIELD SESSION FINAL REPORT • CSM • JUNE 21, 2006

MEDTRONIC

POLARIS SIMULATOR

ABSTRACT

Medtronic Navigation wishes to improve their application testing process. Many of their applications use a localization system, the NDI Polaris camera, and being able to simulate this hardware would improve both the speed and precision of tests, not to mention introduce a way to automate the process. The current testing of these programs requires the camera to be setup and a person to go through an extensive series of maneuvers with different tools. Medtronic wishes to improve this method.

Our program will be limited in scope. We will create an application that is able to simulate one event; this can be built upon in the future to create larger, more complex simulations.

The application works by simulating the responses the Polaris would return. This will be used to simulate the atomic testing sequence, which is a user moving a tool to a certain point in reference to a static reference arc. This will be done in three parts: the recording of a log file, the parsing of this file into more compact, focused files, and finally the playing back of the recorded event.

Table of Contents

Introduction

Medtronic is a world leader in medical technology providing lifelong solutions for people with chronic disease. They offer products, therapies and services that enhance or extend the lives of millions of people. Each year, 5 million patients benefit from Medtronic's technology, used to treat conditions such as diabetes, heart disease, neurological disorders, and vascular illnesses.

Medtronic Navigation is a subsidiary of Medtronic Inc. Medtronic Navigation is the market leader in the field of computer-assisted surgery (CAS); they create software for use with localization cameras such as NDI's Polaris. In recent years, the pace of innovation in CAS has quickened considerably. In response, Medtronic Navigation has developed and delivered new and updated hardware and software solutions to assist with varied surgeries including total joint replacements, minimally invasive spinal surgery, cranial tumor resection, biopsies, functional neurosurgery and Functional Endoscopic Sinus Surgery.

Medtronic Navigation has two releases of their approximately twenty applications each year. Each of these forty releases includes a ten day testing session. This is four hundred man days each year, and at approximately $100 total cost per hour of testing, this comes to $320,000 per year for testing currently. Obviously, this is not a small issue and improving the current process can have a large impact.

Requirements Overview

The goal of this project is to improve the testing process for code that depends on the Polaris localization hardware by mimicking the manual testing with simulation software. Our program is able to copy the manual tests by recording the data sent through the serial port from Polaris, saving it in an appropriate database, and being able to play it back again.

Data will be recorded by allowing the user to start a recording session, do a series of events with the camera, and then save the event. The playback aspect of this project will be script driven and able to initialize the camera, load tool interface files, and run events as described in previous recordings.

This is a small part of a larger project, and as such we have designed our program in such a way as to have it easily continued and expanded upon. In this project we see our scope as emphasized on the recording and saving of events aspects, and have implemented the playback functionality as time allowed.

Requirements
Recording

The recording aspect of our program will be simple. The user must be able to have the program initialize the camera, for any number of wired tools, and record an event. The recording should be able to be paused, resumed, and started over. The recording must create a log file which can be used to create another file(s) for playing back.

Ideally, the recording would be done by siphoning actual data off the serial connection between Medtronic's application and the camera, instead of an independent program to do the recording. However, this might take more time than the project allows and as such will be done as time allows.

Requirements
Saving

The Polaris camera can accept and reply to approximately ten commands each second. With each command being more than ten lines, this is one hundred lines of data each second. For any event more than a couple seconds long the event log will be very large, and as such compression of the data can be important.

The log should be reduced in size by as much as possible. Eliminating unnecessary command sequences, command repetitions, and trivial elements of responses should be considered.

Requirements Playback

This aspect of the program could be the most complicated. Implementing a way of setting up complicated test scripts, which can include several tools and many different events for each tool, should be researched. This simulator has the potential to completely change the way Medtronic tests their application, but a weak customization tool for the test scenarios could severely limit its application for real use.

That said, working within the limited timeframe of this project, we will attempt to have just a simple playback aspect completed. This should be able to recreate one event scenario, this includes initializing the camera, setting up the tools, and simulating the localization responses.

Requirements Implementation

Medtronic Navigation's applications (and most of their computers that would be testing them) work with Linux. However, they are willing to allow us to work in Windows, as long as the code we write is portable enough so as to allow the Windows only parts (e.g. MFC related) to be replaced with their Linux variety.

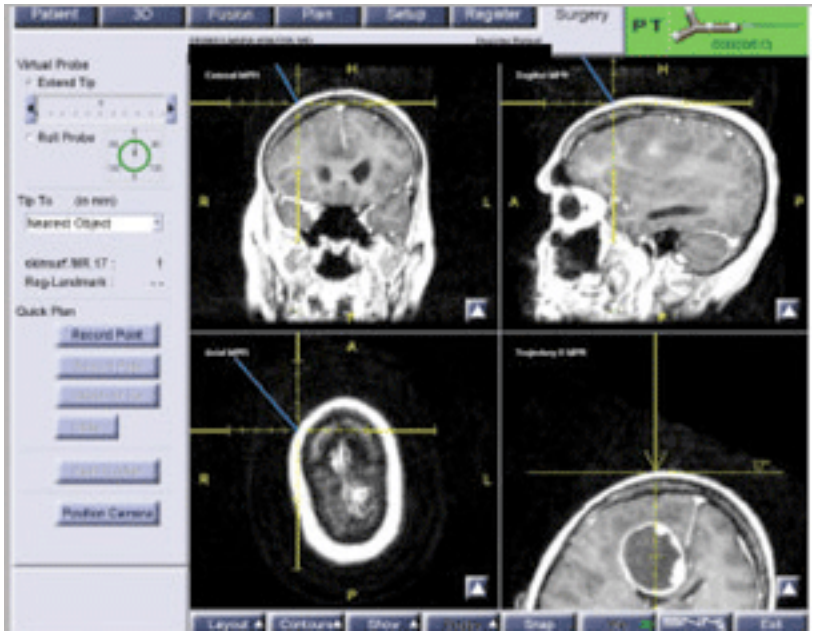
Medtronic Navigation works with C++, and as we feel comfortable in that language, we will work with it for this project.

DESIGN

INTRODUCTION

The three main aspects of our design are recording events, saving these events efficiently, and playing them back in imitation of a camera. Each of these aspects has been designed to work independently, as a base to be built upon, and this modularization allows for each aspect to be developed and expanded without a total rework of the system.

The recording aspect will create a log file of the commands sent to the Polaris system and the responses returned. This log will be parsed by the saving aspect, reducing the size of the file by eliminating unnecessary components. This reduced log of the event can then be played back by the final aspect of the program, simulating the way the Polaris system returns responses after receiving commands. As long as these log files don't change, each part of the project can be expanded without effecting the overall project.



Design Recording

The recorder is most concerned with communicating with the Polaris camera. This is done over a serial connection; we have used a legacy serial port communication class from NDI instead of designing our own. This class is Windows specific, and should be replaced if the program is to be used on another platform.

The main flow for setting up the camera is

1. Camera initialization
    a. Serial Break – Hardware command, camera resets when this is done. When this command completes it is in Setup mode
    b. INIT – Initializes the camera, required for many following commands
    c. ... – Many custodial calls (i.e. VER, SSTAT). Used for creating a descriptive log file and changing camera settings
2. Tool port initialization
    a. PSTAT – can be sent repeatedly, as in the NDI test application. This is used to determine which ports have tools in them
    b. PINIT, PENA for each port with a tool in it – This initializes and enables each port for tracking.
    c. TSTART – This command moves the camera into tracking mode.
3. Event localization
    a. GX – sent nonstop until TSTOP. Returns data about tools location and sensor visibility.
    b. TSTOP – Used to break out of tracking mode and reenter setup mode.

The event localization loop is the longest part of the process; it is where the camera records the position of the tools. The program will save to a log each command and response, which is parsed during saving into files useable for playback.

The log saved by the recorder contains all of the commands and responses the camera was sent and returned. Many of these are trivial (e.g. VER) and can be ignored, others are sent many times in some cases (e.g. PSTAT) and only the response from one is needed for playback purposes, and the most prevalent command by far, GX, returns a response for each port no matter how many ports are active. All of these can be ignored for purposes of playback, and can be cut out.

The initialization sequence is application specific and might need to be saved, but should not be needed for each individual event. Therefore, a master initialization sequence should be recorded and used for all playback scenarios.

The tool initialization sequence is specific to the tools used in the event. The response for each tool should be saved, but no order need be kept. A tool in port 1 when recorded could be played back in port 2 without problem.

The localization command, GX, returns the same number of lines of data every time, no matter how many tools are set up. This means that for events with few number of tools many lines of this command will return “UNOCCUPIED”, and ignored. These responses can be generated at need by the playback loop and an be cut out.

Design Playback

Playback's current state is more rudimentary when compared to its ideal final version than the other two aspects. The current system is to simply choose a camera setup log, tool setup log, and event log, and to run through them from top to bottom. These logs can not be changed after playback is started, and each log's responses will be looped until "special commands" are sent to go to the next log. These commands indicate the camera is changing its state, from camera initialization to port initialization, from port to event, and from event back to port.

Design Summary

The program is split into three aspects : the recorder, saver, and playback. The recorder saves a log of commands sent to the camera and responses received. The saver parses this log file and creates three more condense files, one of which can be ignored/deleted for the most part. Playback will loop through one event log for the current implementation.

Implementation

The project code has been written exclusively in C++. Medtronic put this forth as the most helpful option from their point of view, and as everyone in the group has had experience with this language we felt it would work well. This turned out to be a good decision, as most of the example code we found for serial communication was written in C++.

One program we found was an open source Polaris test program NDI wrote as a test / example application, which they graciously provided it for our use. In it was included a Comm32 class which took care of serial port communication. We have used this class for our own serial communication. It is Windows only, and so if our program is to be compiled for another system this class needs to be replaced with one compatible with that other system.

Medtronic put forth Linux as their operating system of choice. This presented a problem for us, as access to Linux machines would be limited. However, Medtronic was amiable to us working with Windows instead, as long as our code was modularized enough so as to be easily ported to Linux. The item this applies to in our program is the serial communication class, Comm32.

We chose to write the graphical user interface (GUI) in Qt (pronounced as “cute”). The Qt toolkit is a cross platform graphical widget toolkit for the development of GUI computer programs. It uses a signal / slot interface which we used extensively to power our program. Qt works well with C++ and is easily compliable on both Windows and Linux systems.

As for the program itself, our implementation succeeded in recording an event log from the camera and in simulating the event back to a third party test program.

As the project progressed the main focus of the project drifted away from plain recording and saving to recreation of an atomic test scenario. This meant that while recording was completed as originally planned, we spent more time on playback than on saving. The final result is a recorder, playback, but no saving. This is the most trivial of the three aspects though, and should not be as hard to pickup.

Future Direction

If we had more time we would like to expand on all aspects of the program. The recorder could be converted from a program which is used to directly create a log from the camera to a more passive approach, simply saving the commands and responses between the camera and Medtronic's application instead of being the application. The saver could be converted from a simple file parser to a full database controller. Each event could be saved, searched, sorted, split, and combined. Playback could be converted from one event to a play list, script driven, and command line driven. This would make it very powerful for automated regression tests, something currently not feasible for Medtronic.

Conclusions

We have completed the implementation of a test scenario for an atomic test event. This means that our program is able to record and playback one event; an event that would be combined with other, similar, events to create larger, more complex test scenarios. This program should be considered the first step to automating the testing process, as a base to expand to create a system for speeding, expanding, and perhaps replacing the current manual testing practices.

We were able to meet the basic needs of automated testing. We recreated an atomic testing scenario, something which is a base on which the rest of a more powerful system can be built.

APPENDIX
REFERENCES

- Virtual serial port http://www.virtual-serial-port.com
- Polaris Users Manual
- Qt http://www.trolltech.com/
- NDI (Northern Digital Inc.) http://www.ndigital.com/polaris.php
  Comm32 Serial Communication Class from WinPolarisSample
- Gantt http://www.ganttchart.com/
- Medtronic.com

APPENDIX

LIST OF ABBREVIATIONS

CRC::Cyclic Redundancy Check Used to ensure correct data returned from camera

Qt::Not actually abbreviation, pronounced as “cute” Used to create GUI, implements a signal and slot mechanism

GUI::Graphical User Interface How the user interacts with the software

NDI::Northern Digital Inc. Maker of the Polaris camera, the camera Medtronic uses for their localization software

CAS::Computer Assisted Surgery Type of software Medtronic Navigation produces, uses Polaris (and other camera systems) to assist surgeons in surgery by showing the location of tools relative to the patient

APPENDIX

LIST OF COMMANDS & RESPONSES

Serial Break Hardware command, camera resets when this is detected and returns “RESET” if all went well. Camera enters setup mode

INIT Camera enters initialized mode

PSTAT Returns status of each port. If a port has been initialized it should return data about the tool.

PINIT Initializes a port, reads tool data into camera

PENA Enables a port to be tracked

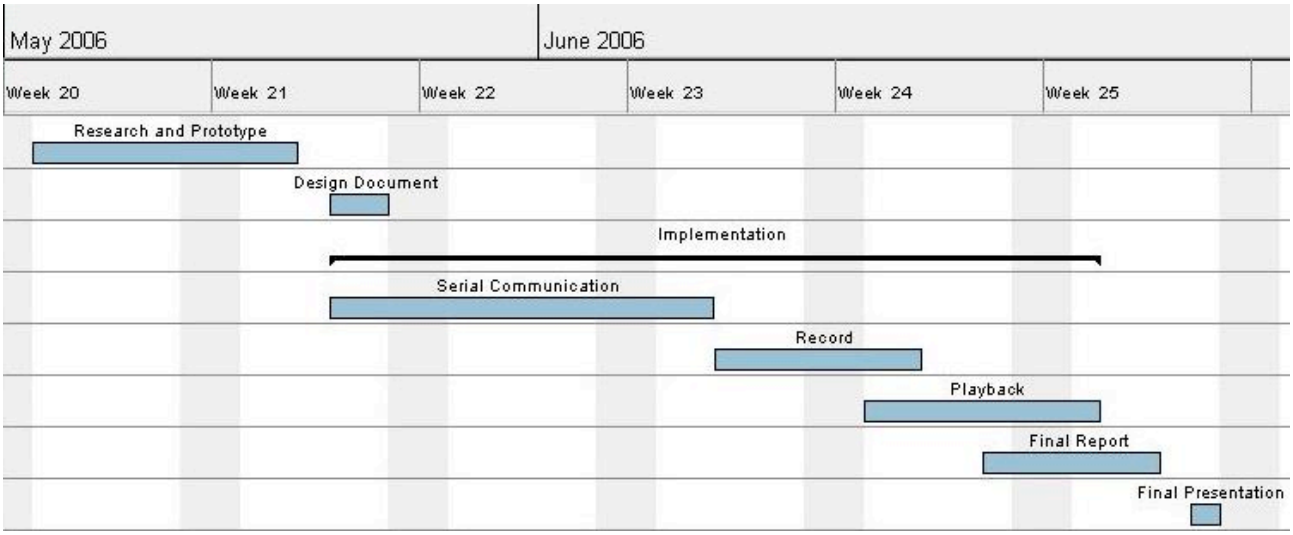
TSTART Camera enters tracking mode

TSTOP Camera exits tracking mode

GX Returns position and transform data from each port. If a tool can not be seen it returns “EMPTY” for that port.

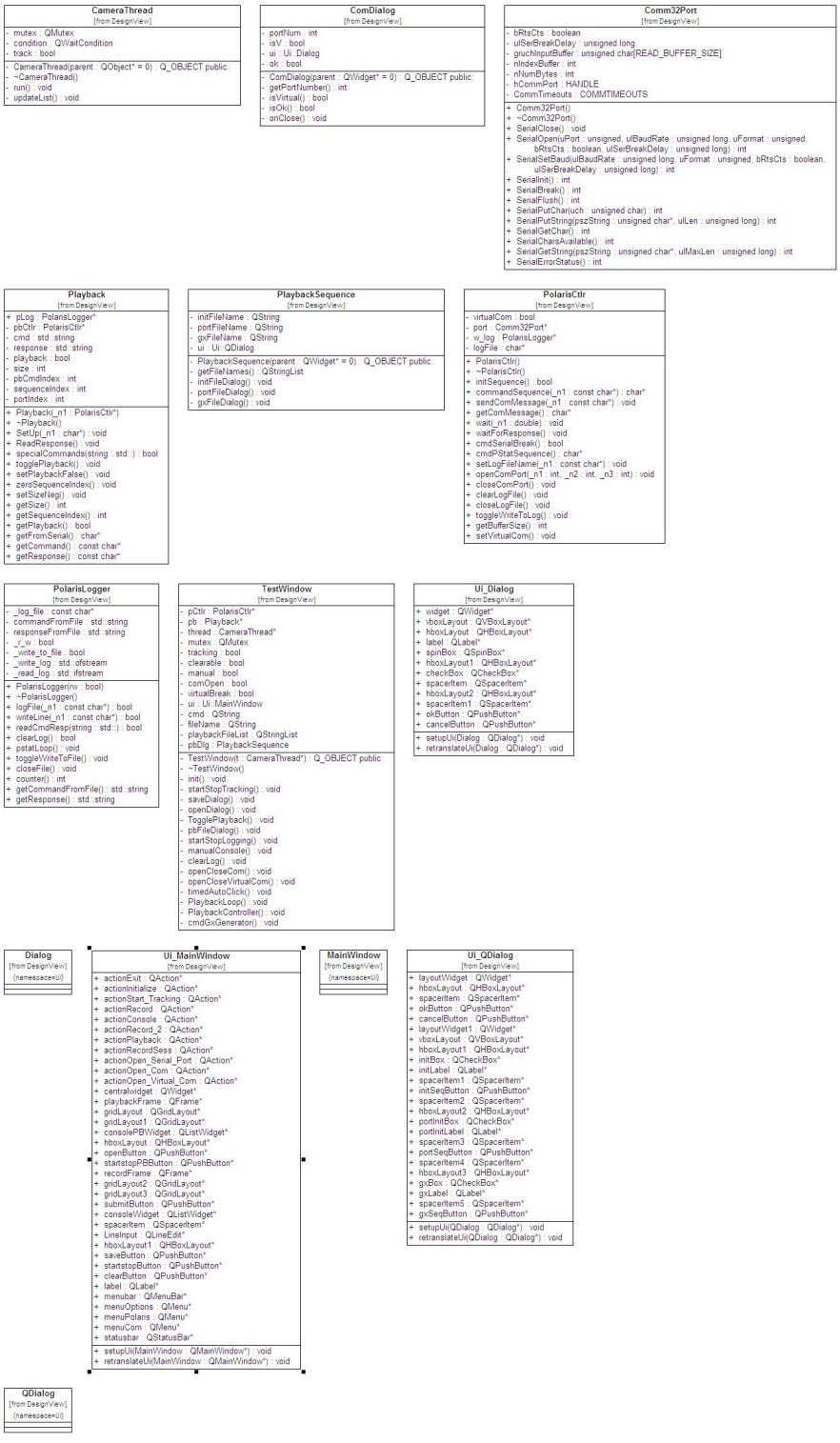
APPENDIX

GANTT TIMELINE



APPENDIX

CLASS STRUCTURE



APPENDIX

DATA FLOW DIAGRAM