VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

AUGMENTED REALITY APPLICATIONS IN EMBEDDED NAVIGATION DEVICES

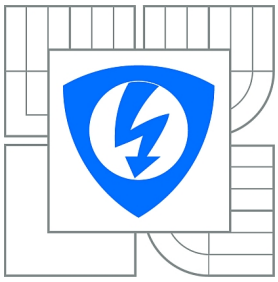
SEMESTRÁLNÍ PRÁCE
SEMESTRAL THESIS

AUTOR PRÁCE MARTIN JAROŠ
AUTHOR

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF RADIO ELECTRONICS

AUGMENTED REALITY APPLICATIONS IN EMBEDDED NAVIGATION DEVICES

AUGMENTED REALITY APPLICATIONS IN EMBEDDED NAVIGATION DEVICES

SEMESTRÁLNÍ PRÁCE

SEMESTRAL THESIS

AUTOR PRÁCE — MARTIN JAROŠ

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. TOMÁŠ FRÝZA, Ph.D.

SUPERVISOR

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Semestrální práce

bakalářský studijní obor
Elektronika a sdělovací technika

Student: Martin Jaroš ***ID:*** 146847
Ročník: 3 ***Akademický rok:*** 2013/2014

NÁZEV TÉMATU:

Augmented reality applications in embedded navigation devices

POKYNY PRO VYPRACOVÁNÍ:

BB2E: Analyze the hardware possibilities of the OMAP platform and design an application to effectively combine captured video data and rendered virtual scene based on navigational data from GPS and INS sensors. Design and create a functional prototype.

BBCE: Examine practical use cases of the proposed navigation device, design applicable user interface.

DOPORUČENÁ LITERATURA:

[1] BIMBER, O.; RASKAR, R. Spatial augmented reality: merging real and virtual worlds. Wellesley: A K Peters, 2005, 369 p. ISBN 15-688-1230-2.

[2] Texas Instruments. OMAP 4460 Multimedia Device [online]. 2012 - [cit. 8. listopadu 2012]. Available: http://www.ti.com/product/omap4460.

Termín zadání: 23.9.2013 ***Termín odevzdání:*** 20.12.2013

Vedoucí práce: doc. Ing. Tomáš Frýza, Ph.D.
Konzultanti semestrální práce:

doc. Ing. Tomáš Kratochvíl, Ph.D.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato práce se zabývá aplikací rozšířené reality v oblasti navigačních zařízení. Popisuje možnosti zpracování videa za účelem projekce virtuální scény pomocí dat získaných satelitním a inerciálním navigačním sybsystémem. Práce klade důraz na využití moderních hardwarových prostředků pro zpracování videa v mikroprocesorech pomocí grafických akcelerátorů. Součástí je návrh aplikace a realizace prototypu.

KLÍČOVÁ SLOVA

Rozšířená realita, satelitní navigace, inerciální měřící jednotka

ABSTRACT

This work deals with application of augmented reality in navigation devices. It describes possibilities of video processing, rendering a virtual scene by using data measured by satellite and inertial navigation subsystems. Special care is taken into account for use of modern graphic accelerator hardware available in microprocessors. Design of the application is supplemented with prototype realization.

KEYWORDS

Augmented reality, satellite navigation, inertial measurement unit

JAROŠ, M. Augmented reality applications in embedded navigation devices. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2014. 50 s. Vedoucí semestrální práce doc. Ing. Tomáš Frýza, Ph.D..

PROHLÁŠENÍ

Prohlašuji, že svou semestrální práci na téma Augmented reality applications in embedded navigation devices jsem vypracoval samostatně pod vedením vedoucího semestrální práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené semestrální práce dále prohlašuji, že v souvislosti s vytvořením této semestrální práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a~jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce doc. Ing. Tomáši Frýzovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé semestrální práce.

Brno ....

(podpis autora)

Contents

List of Figures

List of Tables

Preface

Introduction [1]

1 Augmented reality

1.1 Design goals

Project overview

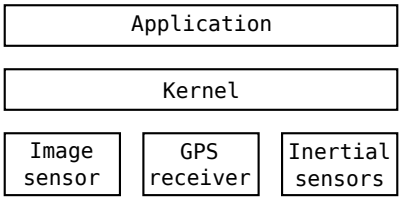


Figure 1: Project overview

1.2 Hardware limitations

Application is designed to run in embedded environments, where power management is very important. While many platforms features multiple symmetrical processor cores - CPUs, application should focus on lowest per-core usage as possible. This can be done by delegating specific tasks to specialized hardware. CPU is specialized in execution of a single thread, integer and bitwise operations, with many branches in its code. With vector and floating point extensions they are also very efficient in computation of difficult mathematical algorithms. However they do not perform well in simple calculations over large amounts of data, where mass parallelization is possible. This is the case in graphics where special graphics processors - GPUs have been deployed. GPU consists of high number (hundreds) of simple cores, which are able to perform operations over large blocks of data. They scale efficiently with the amount of data needed to be processed due to parallelization, however they have problems with nonlinear code with branches. While CPUs have long pipelines for branch optimizations, GPUs cannot employ those, any branch in their code will be extremely inefficient and should be avoided. Graphics chapter focuses on this area. There are also available specialized subsystems designed and optimized for a single purpose. For example video accelerators, capable of video encoding and decoding, image capture systems or peripheral drivers. They will be mentioned in specific chapters.

Developing an application for an embedded device faces a problem, as there are big differences between these devices it is hard to support the hardware and make the application portable. In order to reuse code and reduce application size, libraries are generally used to create an intermediary layer between application and the hardware. However, to provide enough abstraction some sort of operating system has to be used. Operating systems may be real-time, giving applications full control, behaving just like large libraries. This is favorable approach in embedded systems as it allows precise timings. There are many such systems specially tailored for embedded applications like FreeRTOS or proprietary VxWorks. On the other hand, as recent processors improved greatly in power, efficiency and capabilities, it is

possible and quite feasible to run a full featured system like Linux or proprietary Windows CE. Linux kernel is highly portable and configurable, although it does restrict applications from real-time use (Linux RT patches also exist for real-time applications), as all hardware dependent modules which requires full control over the hardware are part of the kernel itself, application does not need to run in real-time at all. Other advantages are free, open and well documented sources, highly standardized and POSIX compliant application interface, large amount of drivers with good manufacturer support. While its disadvantages are very large code base and steep learning curve, which may slow the initial development. Nevertheless Linux kernel has been chosen for the project, more details about its interfaces are in the Linux kernel chapter. While the application is designed to be highly portable depending only on the kernel itself, several devices has been chosen as the reference, they are listed in the hardware chapter.

2 Application

2.1 Linux kernel

Programs running in Linux are divided into two groups, kernel-space and user-space. Only kernel and its runtime modules are allowed to execute in kernel-space, they have physical memory access and use CPU in real-time. All other programs runs as processes in user-space, they have virtual memory access, which means their memory addresses are translated to the physical addresses in the background. In Linux each process runs in a sandbox, isolated from the rest of the system. Processes access memory unique to them, they cannot access memory assigned for other processes nor memory managed by the kernel. They may communicate with the outside environment by several means:

- Arguments and environment variables
- Standard input, output and error output
- Virtual File System
- Signals
- Sockets
- Memory mapping

Each process is ran with several arguments in a specific environment with three default file descriptors. For example running

```
VARIABLE=value ./executable argument1 argument2 <input 1>output 2>error
```

will execute `executable` with environment variable `VARIABLE` of value `value` with two arguments `argument1` and `argument2`. Standard input will be read from file `input` while regular output will be written to file `output` and error output to file `error`. This process may further communicate by accessing files in the Virtual File System, kernel may expose useful process information for example via `procfs` file-system usually mounted at `/proc`. Other types of communication are signals (which may be sent between processes or by kernel) and network sockets. With internal network loop-back device, network style inter process communication is possible using standard protocols (UDP, TCP, ...). Memory mapping is a way to request access to some part of the physical memory.

Process execution is not real-time, but they are assigned restricted processor time by the kernel. They may run in numerous threads, each thread has preemptively scheduled execution. Threads share memory within a process, memory access to these shared resources must done with care to avoid race conditions and data corruption. Kernel provides *mutex* objects to lock threads and avoid simultaneous memory access. Each shared resource should be attached to a *mutex*, which is locked during access to this resource. Thread must not lock *mutex* while still holding lock to this or any other *mutex* in order to avoid dead-locking. Source example on how to use threads is in the appendix A.

Linux kernel has monolithic structure, so all device drivers resides in the kernel-space. From application point of view, this means that all peripheral access must be done through the

standard library and Virtual File System. Individual devices are accessible as device files defined by major and minor number typically located at `/dev`. These files could be created automatically by kernel (`devtmpfs` file-system), by daemon (`udev(8)`), or manually by `mknod(1)`. Complete kernel device model is exported as `sysfs` file-system and typically mounted at `/sys`.

Function name	Access type	Typical usage
`select()`, `poll()`	event	Synchronization, multiplexing, event handling
`ioctl()`	structure	Configuration, register access
`read()`, `write()`	stream	Raw data buffers, byte streams
`mmap()`	block	High throughput data transfers

Table 1: Available functions for working with device file descriptors

For example let's assume a generic peripheral device connected by the I^2C bus. First, to tell kernel there is such a device, the `sysfs` file-system may be used

```
echo $DEVICE_NAME $DEVICE_ADDRESS > /sys/bus/i2c/devices/i2c-1/new_device
```

This should create a special file in `/dev`, which should be opened by `open()` to get a file descriptor for this device. Device driver may export some *ioctl* requests, each request is defined by a number and a structure passed between the application and the kernel. Driver should define requests for controlling the device, maybe accessing its internal registers and configuring a data stream. Each request is called by

```
ioctl(fd, REQNUM, &data);
```

where `fd` is the file descriptor, `REQNUM` is the request number defined in the driver header and `data` is the structure passed to the kernel. This request will be synchronously processed by the kernel and the result stored in the `data` structure. Let's assume this devices has been configured to stream an integer value every second to the application. To synchronize with this timing application may use

```
struct pollfd fds = {fd, POLLIN};
poll(&fds, 1, -1);
```

which will block infinitely until there is a value ready to be read. To actually read it,

```
int buffer[1];
ssize_t num = read(fd, buffer, sizeof(buffer));
```

will copy this value to the buffer. Copying causes performance issues if there are very large amounts of data. To access this data directly without copying them, application has to map physical memory used by the driver. This allows for example direct access to a DMA channel, it should be noted that this memory may still be needed by kernel, so there should be some kind of dynamic access restriction, possibly via *ioctl* requests (this would be driver specific).

2.2 Video subsystem

Video support in Linux kernel is maintained by the LinuxTV project, it implements the `videodev2` kernel module and defines the *V4L2* interface. Modules are part of the mainline kernel at `drivers/media/video/*` with header `linux/videodev2.h`. The core module is enabled by the `VIDEO_V4L2` configuration option, specific device drivers should be enabled by their respective options. V4L2 is the latest revision and is the most widespread video interface throughout Linux, drives are available from most hardware manufactures and usually mainlined or available as patches. The Linux Media Infrastructure API[2] is a well documented interface shared by all devices. It provides abstraction layer for various device implementations, separating the platform details from the applications. Each video device has its device file and is controlled via *ioctl* calls. For streaming standard I/O functions are supported, but the memory mapping is preferred, this allows passing only pointers between the application and the kernel, instead of unnecessary copying the data around.

Name	**Description**
VIDIOC_QUERYCAP	Query device capabilities
VIDIOC_G_FMT	Get the data format
VIDIOC_S_FMT	Set the data format
VIDIOC_REQBUFS	Initiate memory mapping
VIDIOC_QUERYBUF	Query the status of a buffer
VIDIOC_QBUF	Enqueue buffer to the kernel
VIDEOC_DQBUF	Dequeue buffer from the kernel
VIDIOC_STREAMON	Start streaming
VIDIOC_STREAMOFF	Stop streaming

Table 2: V4L2 ioctl calls defined in `linux/videodev2.h`

Application sets the format first, then requests and maps buffers from the kernel. Buffers are exchanged between the kernel and the application. When the buffer is enqueued, it will be available for the kernel to capture data to it. When the buffer is dequeued, kernel will not access the buffer and application may read the data. After all buffer are enqueued application starts the stream. Polling is used to wait for the kernel until it fills the buffer, buffer should not be accessed simultaneously by the kernel and the application. After processing the buffer, application should return it back to the kernel queue. Note that buffers should be properly unmapped by the application after stopping the stream.

Source example for simple video capture is in appendix B. The image format is specified using the little-endian four-character code (FOURCC). V4L2 defines several formats and provides `v4l2_fourcc()` macro to create a format code from four characters. As described later in the

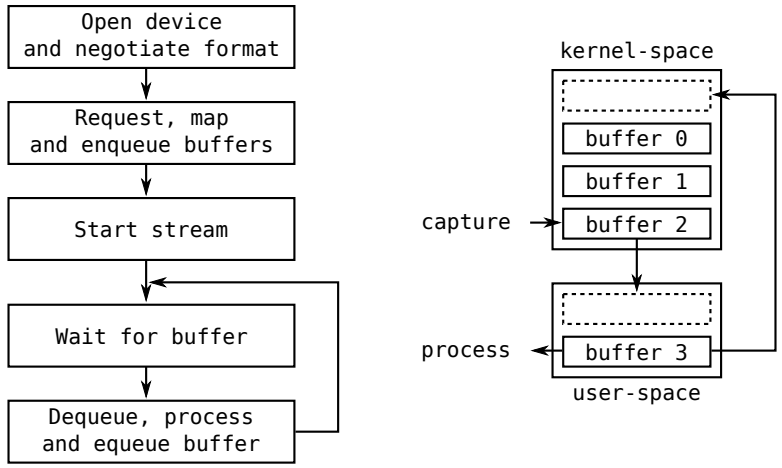


Figure 2: V4L2 capture

graphics subsystem chapter, graphics uses natively the RGB4 format. This format is defined as a single plane with one sample per pixel and four bytes per sample. These bytes represents red, green and blue channel values respectively. Image size is therefore $width \cdot height \cdot 4$ bytes. Many image sensors however support YUV color-space, for example the YU12 format. This one is defined as three planes, the first plane with one luminance sample per pixel and the second and third plane with one chroma sample per four pixels (2 pixels per row, interleaved). Each sample has one byte, this format is also referenced as YUV 4:2:0 and its image size is $width \cdot height \cdot 1.5$ bytes. The luminance and chroma of a pixel is defined as

$$E_Y = W_R \cdot E_R + (1 - W_R - W_B) \cdot E_G + W_B \cdot E_B,$$

$$E_{C_r} = \frac{0.5(E_R - E_Y)}{1 - W_R},$$

$$E_{C_b} = \frac{0.5(E_B - E_Y)}{1 - W_B},$$

where E_R, E_G, E_B are normalized color values and W_R, W_B are their weights. ITU-R Rec. BT.601[3] defines weights as 0.299 and 0.114 respectively, it also defines how they are quantized

$$Y = 219E_Y + 16,$$

$$C_r = 224E_{C_r} + 128,$$

$$C_b = 224E_{C_b} + 128.$$

To calculate R, G, B values from Y, Cr, Cb values, inverse formulas must be used

$$E_Y = \frac{Y - 16}{219},$$

$$E_{C_r} = \frac{C_r - 128}{224},$$

$$E_{C_b} = \frac{C_b - 128}{224},$$

$$E_R = E_Y + 2E_{C_r}(1 - W_R),$$
$$E_G = E_Y - 2E_{C_r}\frac{W_R - {W_R}^2}{W_G} - 2E_{C_b}\frac{W_B - {W_B}^2}{W_G},$$
$$E_B = E_Y + 2E_{C_b}(1 - W_B).$$

It should be noted that not all devices may use the BT.601 recommendation, V4L2 refers to it as `V4L2_COLORSPACE_SMPTE170M` in the `VIDIOC_S_FMT` request structure. Implementation of the YUV to RGB color-space conversion is most efficient on graphics accelerators, such example is included in appendix C. It is written in GLSL for fragment processor, see graphics subsystem chapter for further description.

There is a kernel module `v4l2loopback` which creates a video loop-back device, similar to network loop-back, allowing piping two video applications together. This is very useful not only for testing, but also for implementation of intermediate decoders. GStreamer is a powerful multimedia framework widespread in Linux distributions, composed of a core infrastructure and hundreds of plug-ins. This command will create synthetic RGB4 video stream for the application, useful for testing

```
modprobe v4l2loopback
gst-launch videotestsrc pattern=solid-color foreground-color=0xE0F0E0 ! \
"video/x-raw,format=RGBx,width=800,height=600,framerate=20/1" \
!  v4l2sink device=/dev/video0
```

Texas Instruments distributes a meta package[4] for their OMAP platform featuring all required modules and DSP firmware. This includes kernel modules for *SysLink* inter-chip communication library, *Distributed Codec Engine* library and *ducati* plug-in for GStreamer. With the meta-package installed, it is very easy and efficient to implement mainstream encoded video formats. For example following command will create GStreamer pipeline to receive video payload over a network socket from an IP camera, decode it and push it to the loop-back device for the application. MPEG-4 AVC (H.264) decoder of the IVA 3 is used in this example.

```
modprobe v4l2loopback
gst-launch udpsrc port=5004 caps=\
"application/x-rtp,media=video,payload=96,clock-rate=90000,encoding-name=H264" \
!  rtph264depay !  h264parse !  ducatih264dec !  v4l2sink device=/dev/video0
```

On OMAP4460 this would consume only about 15% of the CPU time as the decoding is done by the IVA 3 video accelerator in parallel to the CPU which only passes pointers around and handles synchronization. Output format is NV12 which is similar to YU12 format described earlier, but there is only one chroma plane with two-byte samples, first byte being the U channel and the second byte the V channel, sampling is same 4:2:0. The YUV to RGB color space conversion must take place here, preferably implemented on the GPU as described above.

Cortex-A9 cores on the OMAP4460 also have the NEON co-processor, capable of vector floating point math. Although not very supported by the GCC C compiler, there are many assembly written libraries implementing coders with the NEON acceleration. For example

the *libjpeg-turbo* library is implementing the *libjpeg* interface. It is useful for USB cameras, as the USB throughput is not high enough for raw high definition video, but is sufficient with JPEG coding (as most USB cameras supports JPEG, but does not support H.264). 1080p JPEG stream decoded with this library via its GStreamer plug-in will consume about 90% of the single CPU core time (note that there are two CPU cores available). However, comparable to the AVC, JPEG encoding will cause visible quality degradation in the raw stream (video looks grainy).

2.3 Graphics subsystem

Graphics stack, OpenGL ES 2.0 [5]

2.4 Inertial measurement subsystem

Application needs to know its spatial orientation for rendering, there three devices which may provide such information, gyroscope, compass (magnetometer) and accelerometer. Hardware details about these devices are in the hardware chapter.

2.4.1 Industrial I/O module

A relatively young kernel module `iio` has been implemented in recent kernels to provide standardized support for sensors and analog converters typically connected by I^2C bus. While many device drivers are still in staging tree, to core module is ready for production code. Subsystem provides device structure mapped in `sysfs`, typically available at `/sys/bus/iio/devices/`. Device are implemented usually on top of the `i2c-dev` driver and registered as `/sys/bus/iio/devices/iio:deviceX`, where X is the device number and the device name may be obtained by

```
cat /sys/bus/iio/devices/iio:deviceX/name
```

There are many possible channels, named by the value type they represents. To read an immediate value, for example from an ADC channel 1

```
cat /sys/bus/iio/devices/iio:deviceX/in_voltage1_raw
cat /sys/bus/iio/devices/iio:deviceX/in_voltage_scale
```

where the result value in volts is $raw \cdot scale$. However, being easy, this is not efficient, buffers have been implemented to stream measured data to the application. Buffer uses device file named after the `iio` device, e.g. `/dev/iio:deviceX`. To stream data through the buffer, driver needs to have control over the timing, triggers have been implemented for this purpose. They are accessible as `/sys/bus/iio/devices/triggerX`, where X is the trigger number and its name may be obtained by

```
cat /sys/bus/iio/devices/triggerX/name
```

Software trigger may be created by

```
echo 1 > /sys/bus/iio/iio_sysfs_trigger/add_trigger
```

and triggered by application

```
echo 1 > /sys/bus/iio/trigger0/trigger_now
```

Name of this trigger is `sysfstrigX`, where X is the trigger number. Hardware triggers are also implemented, both GPIO and timer based triggers. Devices may implement triggers themselves, providing for example the data ready trigger. Device triggers are generally named as `name-devX`, where `name` is device name and `X` is device number. To use trigger with the buffer use

```
echo "triggername" > /sys/bus/iio/devices/iio:deviceX/trigger/current_trigger
```

where `triggername` is the name of the trigger, for example `adc-dev0` will be the device trigger for the ADC. Data are measured in specific channels, they are defined in `/sys/bus/iio/devices/iio:device0/scan_elements`. Channels must be enabled for buffering individually, for example

```
echo 1 > /sys/bus/iio/devices/iio:device0/scan_elements/in_voltage1_en
echo 1 > /sys/bus/iio/devices/iio:device0/scan_elements/in_voltage2_en
```

will enable ADC channels 1 and 2. Buffer itself can be started by

```
echo 256 > /sys/bus/iio/devices/iio:deviceX/buffer/length
echo 1 > /sys/bus/iio/devices/iio:deviceX/buffer/enabled
```

this will start streaming data to the device file. Data are formatted in packets, each packed consists of per-channel values and is terminated by 8 byte time-stamp of the sample. Order of the channels in the buffer can be obtained by

```
cat /sys/bus/iio/devices/iio:device0/scan_elements/in_voltageX_index
```

which reads index of the specified channel. Data format of this channel is

```
cat /sys/bus/iio/devices/iio:device0/scan_elements/in_voltageX_type
```

which reads encoded string, for example `le:u10/16>>0`, where `le` means little-endian, `u` means unsigned, `10` is the number of relevant bits while `16` is the number of actual bits and `0` is the number of right shifts needed.

Following channels are needed by the application:

- `anglvel_x`
- `anglvel_y`
- `anglvel_z`
- `accel_x`
- `accel_y`
- `accel_z`
- `magn_x`
- `magn_y`
- `magn_z`

representing measurements from gyroscope, accelerometer and magnetometer respectively.

2.4.2 DCM algorithm

Equations needed to calculate device attitude have been derived from [6]. Gyroscope measures angular speed around device axes, it offers high differential precision and fast sampling rate, however it suffers slight zero offset error. Device attitude can be obtained simply by integrating measured angular rates, provided that initial attitude is known. The angular rate is defined as

$\overrightarrow{\omega_g} = \frac{\mathrm{d}}{\mathrm{d}t}\overrightarrow{\Phi}_{(t)}$,

so the angular displacement between last two samples is

$[\Phi_x, \Phi_y, \Phi_z] = [\omega_x, \omega_y, \omega_z] \cdot_\Delta t.$

This can be described as a rotation

$$\mathbf{R}_{gyro} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\Phi_x) & -\sin(\Phi_x) \\ 0 & \sin(\Phi_x) & \cos(\Phi_x) \end{bmatrix} \times \begin{bmatrix} \cos(\Phi_y) & 0 & \sin(\Phi_y) \\ 0 & 1 & 0 \\ -\sin(\Phi_y) & 0 & \cos(\Phi_y) \end{bmatrix} \times \begin{bmatrix} \cos(\Phi_z) & -\sin(\Phi_y) & 0 \\ \sin(\Phi_y) & \cos(\Phi_y) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

With ${}_\Delta t$ close to zero a small-angle approximation may be used to simplify $\cos(x) = 1$, $\sin(x) = x$

$$\mathbf{R}_{gyro} \doteq \begin{bmatrix} 1 & -\Phi_z & \Phi_y \\ \Phi_x\Phi_y + \Phi_z & 1 - \Phi_x\Phi_y\Phi_z & -\Phi_x \\ \Phi_x\Phi_z - \Phi_y & \Phi_x + \Phi_y\Phi_z & 1 \end{bmatrix}.$$

Let's define the directional cosine matrix describing device attitude

$$\mathbf{DCM} = \begin{bmatrix} \widehat{\mathbf{I}} \cdot \widehat{\mathbf{x}} & \widehat{\mathbf{I}} \cdot \widehat{\mathbf{y}} & \widehat{\mathbf{I}} \cdot \widehat{\mathbf{z}} \\ \widehat{\mathbf{J}} \cdot \widehat{\mathbf{x}} & \widehat{\mathbf{J}} \cdot \widehat{\mathbf{y}} & \widehat{\mathbf{J}} \cdot \widehat{\mathbf{z}} \\ \widehat{\mathbf{K}} \cdot \widehat{\mathbf{x}} & \widehat{\mathbf{K}} \cdot \widehat{\mathbf{y}} & \widehat{\mathbf{K}} \cdot \widehat{\mathbf{z}} \end{bmatrix} = \begin{bmatrix} \widehat{\mathbf{I}}_{xyz} \\ \widehat{\mathbf{J}}_{xyz} \\ \widehat{\mathbf{K}}_{xyz} \end{bmatrix},$$

where $\widehat{\mathbf{I}}$ points to the north, $\widehat{\mathbf{J}}$ points to the east, $\widehat{\mathbf{K}}$ points to the ground and therefore $\widehat{\mathbf{I}} = \widehat{\mathbf{J}} \times \widehat{\mathbf{K}}$. Roll, pitch and yaw angels in this matrix are

$\gamma = -\arctan_2\left(\frac{\mathbf{DCM}_{23}}{\mathbf{DCM}_{33}}\right)$,

$\beta = \arcsin(\mathbf{DCM}_{13})$,

$\alpha = -\arctan_2\left(\frac{\mathbf{DCM}_{12}}{\mathbf{DCM}_{11}}\right)$.

DCM can be computed by applying consecutive rotations over time

$\mathbf{DCM}_{(t)} = \mathbf{R}_{gyro(t)} \times \mathbf{DCM}_{(t-1)}$.

If the sampling rate is high enough (over 1kHz at least), this method is very accurate and has good dynamics over short periods of time, but in longer runs errors integrated during processing will cause serious drift (both numerical errors and zero offset errors). To mitigate these problems accelerometer and compass has to be used to provide the initial attitude and to fix the drift over time. Accelerometer measures external mechanical forces applied to the device together with gravitational force. However precision of these devices are generally

worse and they have slower sampling rates. If there are no extern forces, it will measure the gravitational vector directly, thus providing the third row of the DCM

$$\overrightarrow{\mathbf{a}}_{acc} = g\ \widehat{\mathbf{K}}_{xyz} + \frac{\overrightarrow{\mathbf{F}}}{m},$$

$$\overrightarrow{\mathbf{F}} = 0\ \rightarrow\ \widehat{\mathbf{K}}_{xyz} = \frac{\overrightarrow{\mathbf{a}}_{acc}}{\left|\overrightarrow{\mathbf{a}}_{acc}\right|}.$$

When there is an external force $\overrightarrow{\mathbf{F}}$ applied, which is not parallel and has significant magnitude relative to gravitational force $m\,g\,\widehat{\mathbf{K}}_{xyz}$, measurements will degrade rapidly reaching singularity during the free fall ($\left|\overrightarrow{\mathbf{a}}_{acc}\right| = 0$). This error may be corrected by using device speed measured by satellite navigation system with high sample rate (over 10Hz)

$$\widehat{\mathbf{K}}_{xyz} = \frac{\overrightarrow{\mathbf{a}}_{acc} - \frac{\mathrm{d}}{\mathrm{d}t}\overrightarrow{\mathbf{v}}_{GPS}}{g}.$$

Magnetometer has similar properties, it measures magnetic flux density of the field the device is within. This should ideally result in a vector pointing to the north, therefore providing the first row of the DCM

$$\widehat{\mathbf{I}}_{xyz} = \frac{\overrightarrow{\mathbf{B}}_{corr}}{\left|\overrightarrow{\mathbf{B}}_{corr}\right|}.$$

Magnetometers have even slower sampling rates and far worse precision as the Earth field is distorted by nearby metal objects. This magnetic deviation can be divided into hard-iron and soft-iron effects. Hard-iron distortion is caused by materials that produces magnetic field, that is added to the Earth magnetic field. Vector of this field can be subtracted to compensate this error

$$\overrightarrow{\mathbf{B}}_{corr1} = \overrightarrow{\mathbf{B}}_{mag} - \tfrac{1}{2}\left[\min(B_x) + \max(B_x), \min(B_y) + \max(B_y), \min(B_z) + \max(B_z)\right].$$

The soft-iron distortion is caused by soft magnetic materials, which reshapes the field in a way that is not simply additive. It may be observed as an ellipse when the device is rotated around and the measured values are plotted. Compensating for these effects is involves remapping this ellipse back to the sphere. This is computation intensive and as soft-iron effects are usually weak (up to few degrees), it may be omitted.

Further more the magnetic field of the Earth itself does not point to the geographic north, but is rotated by an angle specific to the location on the Earth surface. Magnetic inclination is the vertical portion of this rotation causing magnetic vector to incline to the ground, it may be fixed by using measurements from the accelerometer to make the magnetic vector perpendicular to the gravitational vector

$$\overrightarrow{\mathbf{B}}_{corr2} = \widehat{\mathbf{K}}_{xyz} \times \overrightarrow{\mathbf{B}}_{mag} \times \widehat{\mathbf{K}}_{xyz}.$$

Magnetic declination (sometimes referred as magnetic variation) is the horizontal portion of this rotation and is sometimes provided by the satellite navigation systems. To correct for this error, measured values have to be rotated by the inverse angle

$$\overrightarrow{\mathbf{B}}_{corr3} = \overrightarrow{\mathbf{B}}_{mag} \begin{bmatrix} \cos(var) & \sin(var) \\ -\sin(var) & \cos(var) \end{bmatrix}.$$

By combination of the corrected results from accelerometer and magnetometer complete DCM can be calculated. Weighted average should be used, in real-time this yields

$$\mathbf{DCM}_{(t)} = W_{gyro}\ (\mathbf{R}_{gyro} \times \mathbf{DCM}_{(t-1)}) + (1 - W_{gyro}) \begin{bmatrix} \widehat{\mathbf{I}}_{xyz} \\ \widehat{\mathbf{K}}_{xyz} \times \widehat{\mathbf{I}}_{xyz} \\ \widehat{\mathbf{K}}_{xyz} \end{bmatrix},$$

where $\widehat{\mathbf{I}}_{xyz}$ and $\widehat{\mathbf{K}}_{xyz}$ are calculated from magnetometer and accelerometer measurements. W_{gyro} is the weight of the gyroscope measurement, it must be estimated by trial and error to mitigate its drift but not add too much noise.

2.5 Satellite navigation subsystem

TTY module, stty, socat, GPS [7], NMEA 0183 [8]

3 Hardware

OMAP4460 application processor[9]

- two ARM Cortex-A9 SMP general-purpose processors
- IVA 3 video accelerator, 1080p capable
- image signal processor, 20MP capable
- SGX540 3D graphics accelerator, OpenGL ES 2.0 compatible
- HDMI v1.3 video output

MPU-9150 motion tracking device[10]

- embedded MPU-6050 3-axis gyroscope and accelerometer
- embedded AK8975 3-axis digital compass
- fully programmable, I^2C interface

OV5640 image sensor

- 1080p, 5MP resolution
- raw RGB or YUV output

4 Conclusion

Conclusion

References

[1] BIMBER, Oliver and RASKAR, Ramesh. *Spatial augmented reality: merging real and virtual worlds*. Wellesley: A K Peters, 2005. ISBN 15-688-1230-2.

[2] LinuxTV Developers. Linux Media Infrastructure API. [online]. 2012. [Accessed 20 November 2013]. Available from: http://linuxtv.org/downloads/v4l-dvb-apis

[3] Consultative Committee on International Radio. Recommendation BT.601. [online]. 2011. [Accessed 20 November 2013]. Available from: http://www.itu.int/rec/R-REC-BT.601/en

[4] TI OMAP Developers. TI OMAP Trunk PPA. [online]. 2013. [Accessed 20 November 2013]. Available from: https://launchpad.net/~tiomap-dev/+archive/omap-trunk

[5] Khronos Group. OpenGL ES 2.x - for Programmable Hardware. [online]. 2013. [Accessed 20 November 2013]. Available from: http://www.khronos.org/opengles/2_X

[6] JAZAR, Reza N. *Theory of applied robotics: kinematics, dynamics, and control*. 2nd ed. New York: Springer, 2010. ISBN 978-1-4419-1749-2.

[7] KENNEDY, Melita. *Understanding map projections*. Redlands: ESRI, 2000. ISBN 15-894-8003-1.

[8] National Marine Electronics Association. NMEA 0183. [online]. 2008. [Accessed 20 November 2013]. Available from: http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp

[9] Texas Instruments. OMAP 4460 Multimedia Device. [online]. 2012. [Accessed 20 November 2013]. Available from: http://www.ti.com/product/omap4460

[10] InvenSense. MPU-9150 Nine-axis MEMS MotionTracking™ Device. [online]. 2013. [Accessed 20 November 2013]. Available from: http://www.invensense.com/mems/gyro/mpu9150.html

List of Annexes

A Threads example

In this example two threads share standard input and output, access is restricted by *mutex* so only one thread may use the shared resource at any time.

```c
#include <stdio.h>
#include <pthread.h>

void *worker(void *arg)
{
    static int counter = 1;
    pthread_mutex_t *mutex = (pthread_mutex_t*)arg;
    char *buffer;

    // Lock mutex to restrict access to stdin and stdout
    pthread_mutex_lock(mutex);
    printf("This is worker %d, enter something: ", counter++);
    scanf("%ms", &buffer);
    pthread_mutex_unlock(mutex);

    return (void*)buffer;
}

int main()
{
    pthread_mutex_t mutex;
    pthread_t thread1, thread2;
    char *retval1, *retval2;

    // Initialize two threads with shared mutex, use default parameters
    pthread_mutex_init(&mutex, NULL);
    pthread_create(&thread1, NULL, worker, (void*)&mutex);
    pthread_create(&thread2, NULL, worker, (void*)&mutex);

    // Wait for both threads to finish and display results
    pthread_join(thread1, (void**)&retval1);
    pthread_join(thread2, (void**)&retval2);
    printf("Thread 1 returned with `%s`.\n", retval1);
    printf("Thread 2 returned with `%s`.\n", retval2);

    pthread_mutex_destroy(&mutex);
    return 0;
}
```

B Video capture example

In this example video device is configured to capture frames using memory mapping. These frames are dumped to standard output, instead of further processing.

```
#include <fcntl.h>
#include <unistd.h>
#include <poll.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <linux/videodev2.h>

int main()
{
    // Open device
    int fd = open("/dev/video0", O_RDWR | O_NONBLOCK);

    // Set video format
    struct v4l2_format format =
    {
        .type = V4L2_BUF_TYPE_VIDEO_CAPTURE,
        .fmt =
        {
            .pix =
            {
                .width = 320,
                .height = 240,
                .pixelformat = V4L2_PIX_FMT_RGB32,
                .field = V4L2_FIELD_NONE,
                .colorspace = V4L2_COLORSPACE_SMPTE170M,
            },
        },
    };
    ioctl(fd, VIDIOC_S_FMT, &format);

    // Request buffers
    struct v4l2_requestbuffers requestbuffers =
    {
        .type = V4L2_BUF_TYPE_VIDEO_CAPTURE,
        .memory = V4L2_MEMORY_MMAP,
        .count = 4,
    };
    ioctl(fd, VIDIOC_REQBUFS, &requestbuffers);
    void *pbuffers[requestbuffers.count];
```

```
40
41      // Map and enqueue buffers
42      int i;
43      for(i = 0; i < requestbuffers.count; i++)
44      {
45          struct v4l2_buffer buffer =
46          {
47              .type = V4L2_BUF_TYPE_VIDEO_CAPTURE,
48              .memory = V4L2_MEMORY_MMAP,
49              .index = i,
50          };
51          ioctl(fd, VIDIOC_QUERYBUF, &buffer);
52          pbuffers[i] = mmap(NULL, buffer.length,
53                             PROT_READ | PROT_WRITE, MAP_SHARED,
54                             fd, buffer.m.offset);
55          ioctl(fd, VIDIOC_QBUF, &buffer);
56      }
57
58      // Start stream
59      enum v4l2_buf_type buf_type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
60      ioctl(fd, VIDIOC_STREAMON, &buf_type);
61
62      while(1)
63      {
64          // Synchronize
65          struct pollfd fds =
66          {
67              .fd = fd,
68              .events = POLLIN
69          };
70          poll(&fds, 1, -1);
71
72          // Dump buffer to stdout
73          struct v4l2_buffer buffer =
74          {
75              .type = V4L2_BUF_TYPE_VIDEO_CAPTURE,
76              .memory = V4L2_MEMORY_MMAP,
77          };
78          ioctl(fd, VIDIOC_DQBUF, &buffer);
79          write(1, pbuffers[buffer.index], buffer.bytesused);
80          ioctl(fd, VIDIOC_QBUF, &buffer);
81      }
82  }
```

C Colorspace conversion example

In this example RGB to YUV color-space conversion is implemented in fragment shader. Each input channel has its own texturing unit, texture coordinates are divided by sub-sampling factor 4:2:0.

```
uniform sampler2D texY, texU, texV;
varying vec2 texCoord;

void main()
{
    float y = texture2D(texY, texCoord).a * 1.1644 - 0.062745;
    float u = texture2D(texU, texCoord / 2).a - 0.5;
    float v = texture2D(texV, texCoord / 2).a - 0.5;

    gl_FragColor = vec4(
        y + 1.596 * v,
        y - 0.39176 * v - 0.81297 * u,
        y + 2.0172 * u,
        1.0);
}
```