High Speed Camera & IMUs on Mobile Devices

Instructor - Simon Lucey

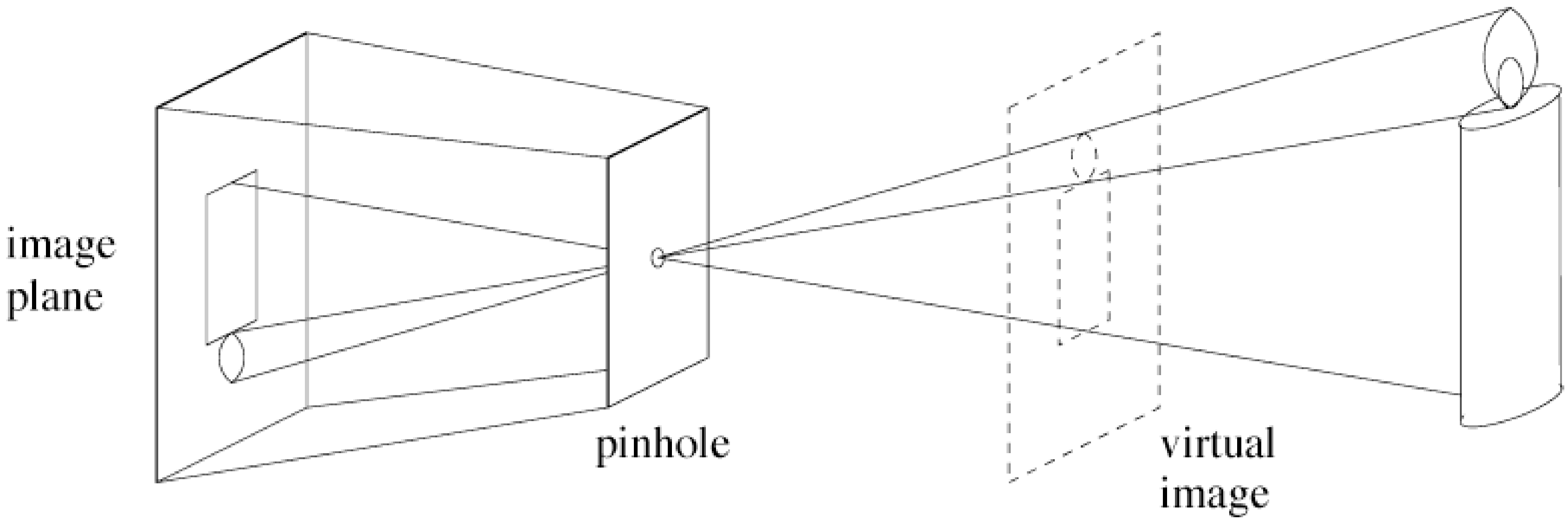
16-623 - Designing Computer Vision Apps

Carnegie Mellon
THE ROBOTICS INSTITUTE

Today

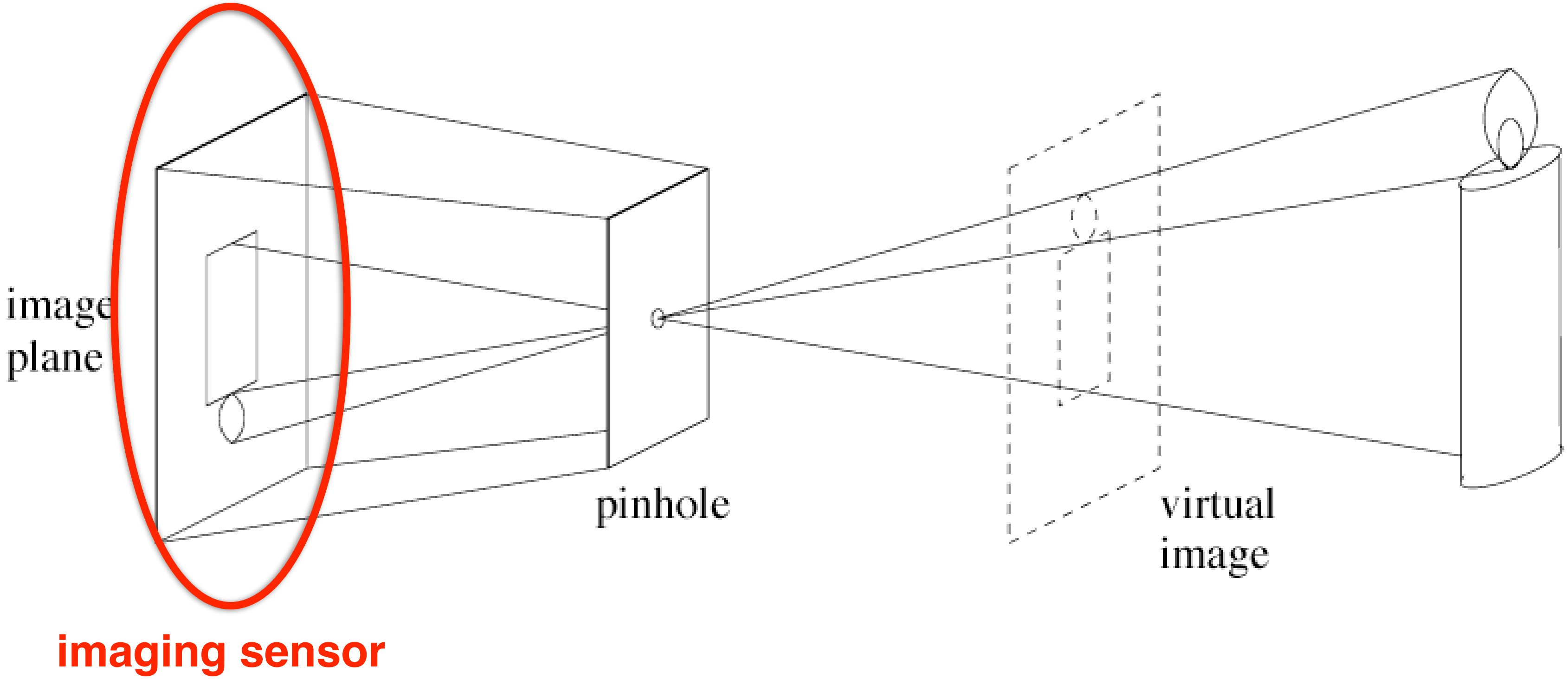
- CCD vs CMOS cameras.
- Rolling Shutter Epipolar Geometry
- Inertial Measurement Units (IMU)

Pinhole Camera



(Taken from Forsyth & Ponce)

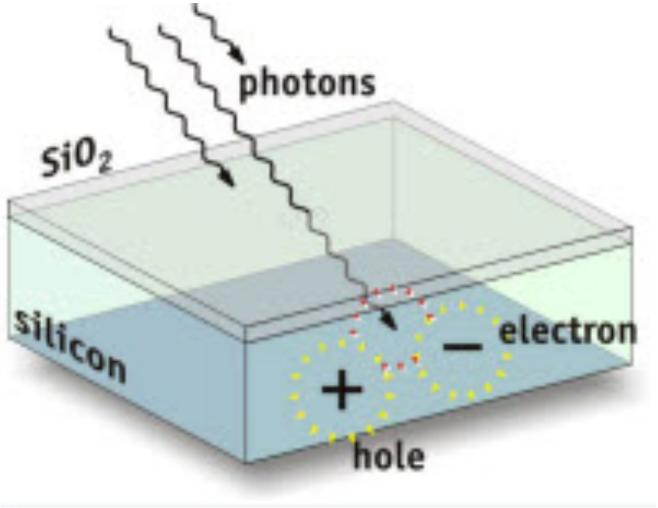
Pinhole Camera



(Taken from Forsyth & Ponce)

Digital Cameras

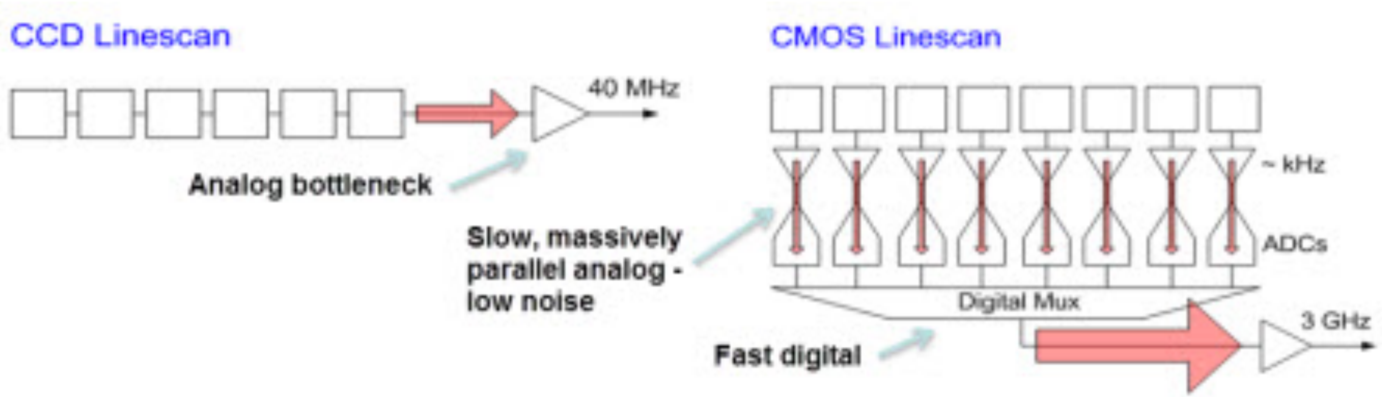
- All digital cameras rely on the photoelectric effect to create electrical signal from light.
- CCD (charge coupled device) and CMOS (complementary metal oxide semiconductor) are the two most common image sensors found in digital cameras.
- Both invented in the late 60s early 70s.



(Taken from https://www.teledynedalsa.com/imaging/knowledge-center/appnotes/ccd-vs-cmos/)

CCD versus CMOS

- CMOS and CCD imagers differ in the way that signals are converted from signal charge.



- CMOS imagers are inherently more parallel than CCDs.
- Consequently, high speed CMOS imagers can be designed to have much lower noise than high speed CCDs.

(Taken from https://www.teledynedalsa.com/imaging/knowledge-center/appnotes/ccd-vs-cmos/)

CCD versus CMOS

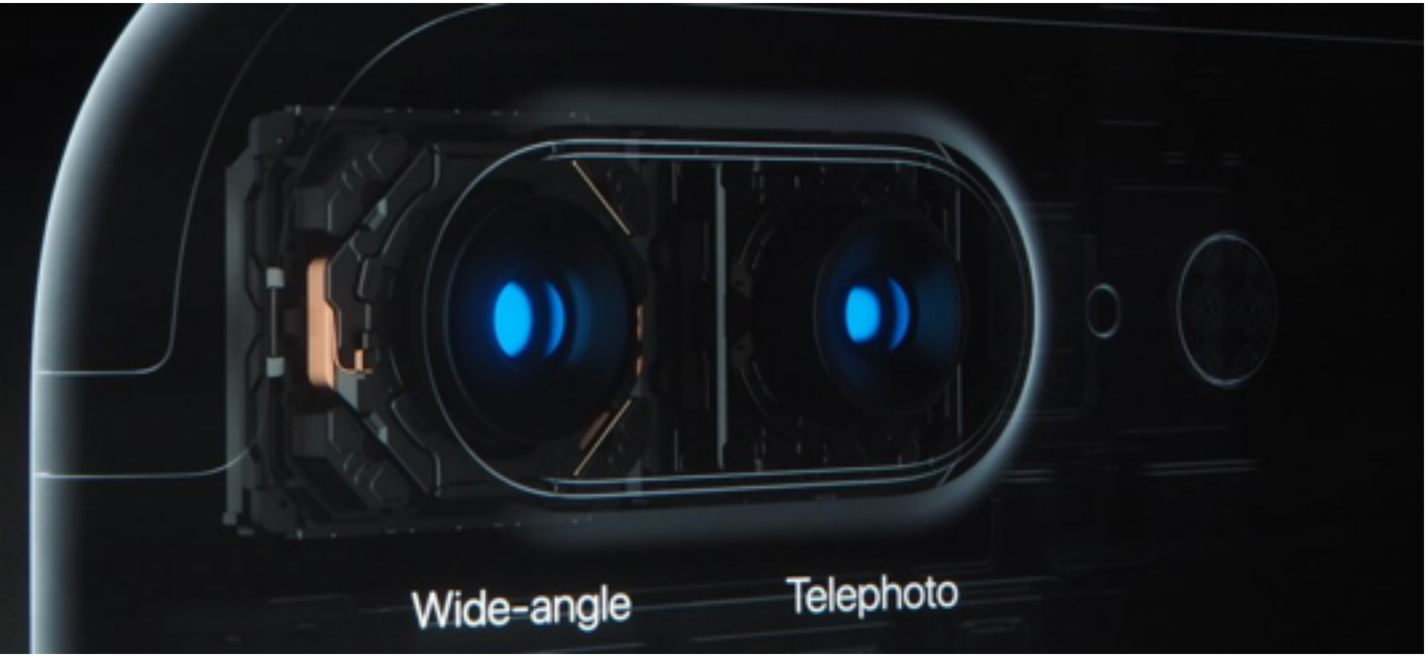
- CCD used to be the image sensor of choice as it gave far superior images with the fabrication technology available.
- CMOS was of interest with the the advent of mobile phones.
  - CMOS promised lower power consumption.
  - lowered fabrication costs (reuse mainstream logic and memory device fabrication).
- An enormous amount of investment was made to develop and fine tune CMOS imagers.
- As a result we witnessed great improvements in image quality, even as pixel sizes shrank.
- In the case of high volume consumer area imagers, CMOS imagers outperform CCDs based on almost every performance parameter.

(Taken from https://www.teledynedalsa.com/imaging/knowledge-center/appnotes/ccd-vs-cmos/)

Taken from: http://9to5mac.com/2014/09/23/iphone-6-camera-compared-to-all-previous-iphones-gallery/

New Developments - iPhone 7

- Apple just released the iPhone 7 with new dual lens camera.
- Rumored that advances in the camera are based on the 2015 acquisition of Linx (Israeli startup).
- Image quality “closest” attempt yet to DSLR on mobile device.

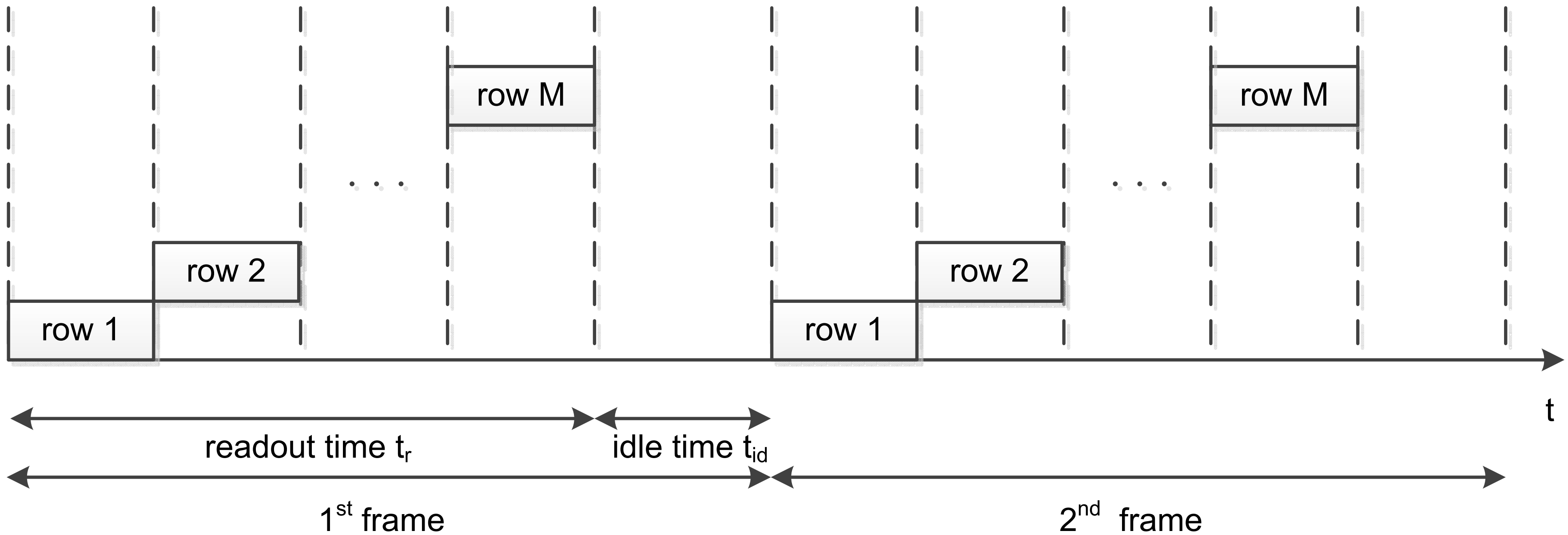


Taken from: http://vrscout.com/news/apple-duel-camera-iphone-for-augmented-reality/

Today

- CCD vs CMOS cameras.
- Rolling Shutter Epipolar Geometry
- Inertial Measurement Units (IMU)

Rolling Shutter Effect

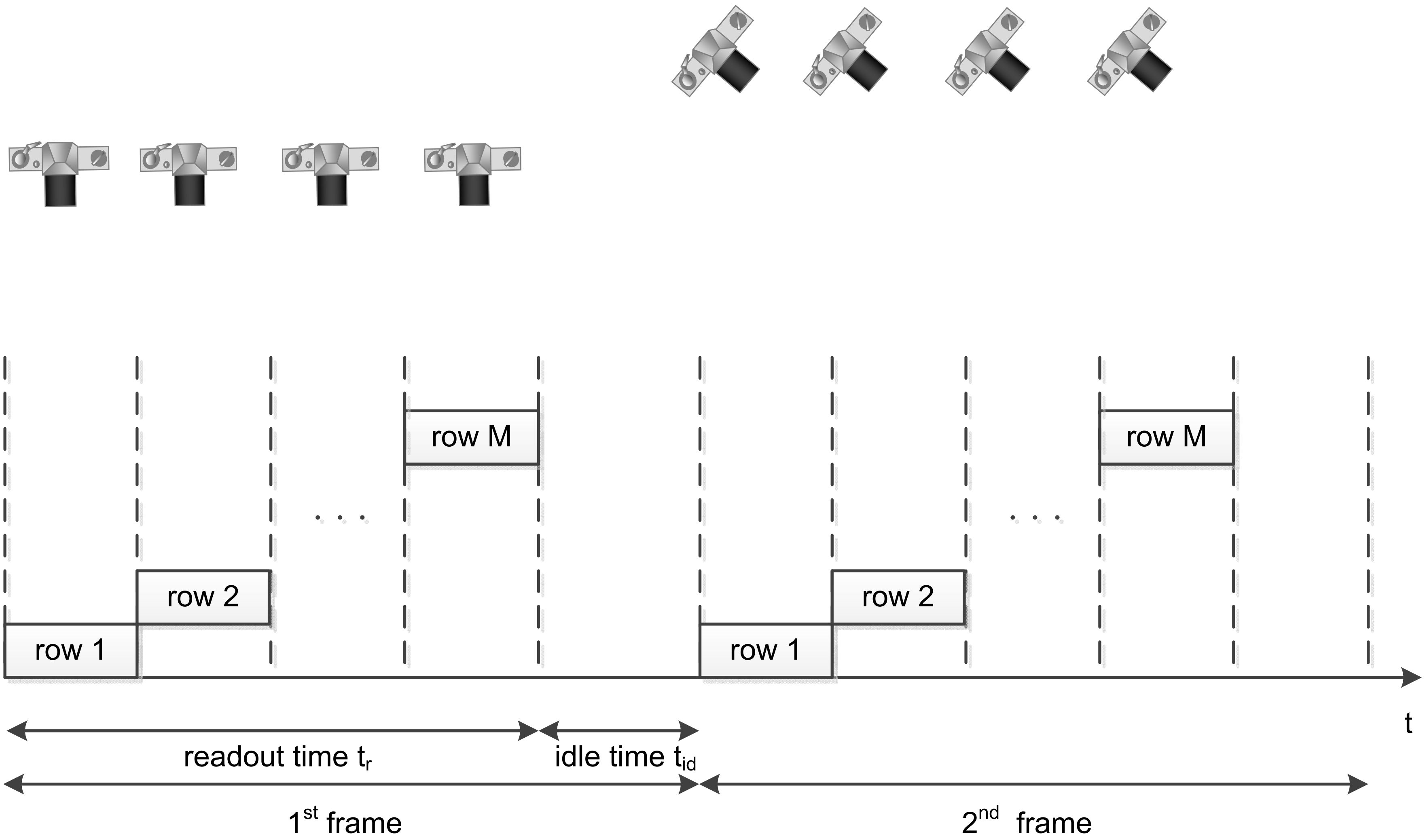


Rolling shutter cameras sequentially expose rows.

$$t_r + t_{id} = \frac{1}{\text{frames per second}}$$

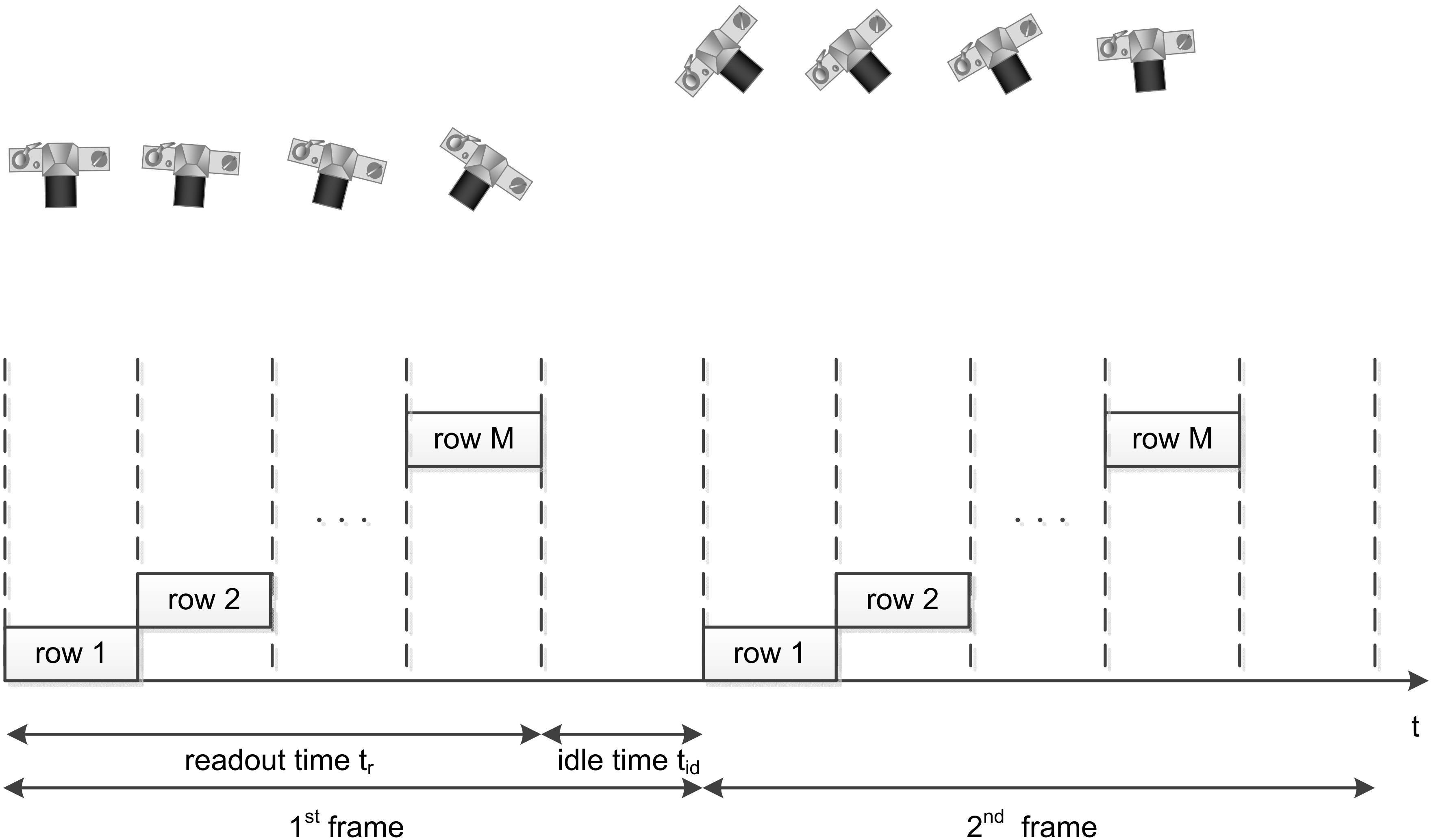
Taken from: Jia and Evans "Probabilistic 3-D Motion Estimation for Rolling Shutter Video Rectification from Visual and Inertial Measurements" MMSP 2012.

Global versus Rolling Shutter



Taken from: Jia and Evans "Probabilistic 3-D Motion Estimation for Rolling Shutter Video Rectification from Visual and Inertial Measurements" MMSP 2012.

Global versus Rolling Shutter



Taken from: Jia and Evans "Probabilistic 3-D Motion Estimation for Rolling Shutter Video Rectification from Visual and Inertial Measurements" MMSP 2012.

Rolling-Shutter Effect

- A drawback to CMOS sensors is the “rolling-shutter effect”.
- CMOS captures images by scanning one line of the frame at a time.
- If anything is moving fast, then it will lead to weird distortions in still photos, and to rather odd effects in video.
- Check out the following video taken with the iPhone 4 CCD camera.
- CCD-based cameras often use a “global” shutter to circumvent this problem.

Taken from: http://www.wired.com/2011/07/iphones-rolling-shutter-captures-amazing-slo-mo-guitar-string-vibrations/

Rolling-Shutter Effect

- A drawback to CMOS sensors is the “rolling-shutter effect”.
- CMOS captures images by scanning one line of the frame at a time.
- If anything is moving fast, then it will lead to weird distortions in still photos, and to rather odd effects in video.
- Check out the following video taken with the iPhone 4 CCD camera.
- CCD-based cameras often use a “global” shutter to circumvent this problem.

Taken from: http://www.wired.com/2011/07/iphones-rolling-shutter-captures-amazing-slo-mo-guitar-string-vibrations/

Rolling Shutter Effect = “Aliasing”

- Rolling Shutter Effect is an example of a broader phenomena regularly studied in Signal Processing called “Aliasing”.
- Common phenomenon
  - Wagon wheels rolling the wrong way in movies.

Rolling Shutter Effect = “Aliasing”

- Rolling Shutter Effect is an example of a broader phenomena regularly studied in Signal Processing called “Aliasing”.
- Common phenomenon
  - Wagon wheels rolling the wrong way in movies.

Rectifying Rolling Shutter

- What do you think the camera motion was here?

Taken from: Hanning et al. "Stabilizing Cell Phone Video using Inertial Measurement Sensors" in ICCV 2011 Workshop.

High-Frame Rate Cameras

- Another way around this is to create higher-frame rate cameras.
- Increasingly seeing faster and faster CMOS cameras.
- Opening up other exciting opportunities in computer vision.
- However, really fast motions still need an understanding of the rolling shutter effect.

High-Frame Rate Cameras

- Another way around this is to create higher-frame rate cameras.
- Increasingly seeing faster and faster CMOS cameras.
- Opening up other exciting opportunities in computer vision.
- However, really fast motions still need an understanding of the rolling shutter effect.

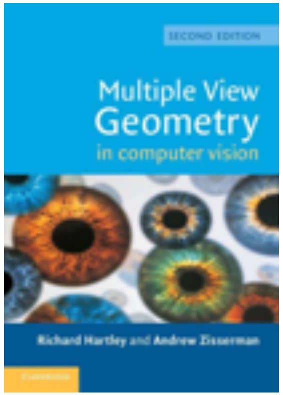
Rectifying Rolling Shutter

- Result from rectification,

Taken from: Hanning et al. “Stabilizing Cell Phone Video using Inertial Measurement Sensors” in ICCV 2011 Workshop.

Reminder: Cheat Sheet

Description	Hartley & Zisserman	Prince
3D Point	$\mathbf{X}$	$\mathbf{w}$
2D Point	$\mathbf{x}$	$\mathbf{x}$
Rotation matrix	$\mathbf{R}$	$\boldsymbol{\Omega}$
Intrinsics matrix	$\mathbf{K}$	$\boldsymbol{\Lambda}$
Homography matrix	$\mathbf{H}$	$\boldsymbol{\Phi}$
translation vector	$\mathbf{t}$	$\boldsymbol{\tau}$



Reminder: The Essential Matrix

First camera:

$$\lambda_1 \tilde{\mathbf{x}}_1 = \mathbf{w}$$

Second camera:

$$\lambda_2 \tilde{\mathbf{x}}_2 = \boldsymbol{\Omega}\mathbf{w} + \boldsymbol{\tau}$$

Substituting:

$$\lambda_2 \tilde{\mathbf{x}}_2 = \lambda_1 \boldsymbol{\Omega} \tilde{\mathbf{x}}_1 + \boldsymbol{\tau}$$

This is a mathematical relationship between the points in the two images, but it's not in the most convenient form.

Reminder: The Essential Matrix

$$\lambda_2 \tilde{\mathbf{x}}_2 = \lambda_1 \boldsymbol{\Omega} \tilde{\mathbf{x}}_1 + \boldsymbol{\tau}$$

$$\lambda_2 \boldsymbol{\tau} \times \tilde{\mathbf{x}}_2 = \lambda_1 \boldsymbol{\tau} \times \boldsymbol{\Omega} \tilde{\mathbf{x}}_1$$

$$\tilde{\mathbf{x}}_2^T \boldsymbol{\tau} \times \boldsymbol{\Omega} \tilde{\mathbf{x}}_1 = 0$$

Adapted from: Computer vision: models, learning and inference. Simon J.D. Prince

Reminder: The Essential Matrix

$$\tilde{\mathbf{x}}_2^T \boldsymbol{\tau} \times \boldsymbol{\Omega}\tilde{\mathbf{x}}_1 = 0$$

The cross product term can be expressed as a matrix

$$\boldsymbol{\tau}_\times = \begin{bmatrix} 0 & -\tau_z & \tau_y \\ \tau_z & 0 & -\tau_x \\ -\tau_y & \tau_x & 0 \end{bmatrix}$$

Defining:

$$\mathbf{E} = \boldsymbol{\tau}_\times \boldsymbol{\Omega}$$

We now have the essential matrix relation

$$\tilde{\mathbf{x}}_2^T \mathbf{E} \tilde{\mathbf{x}}_1 = 0$$

Epipolar Geometry for Rolling Shutter

- Recently Dai et al. (2016) developed Generalized Epipolar Geometry for Rolling Shutter Camera.
- Assuming linear rolling shutter,

$$\lambda_1 \tilde{\mathbf{x}}_1 = \mathbf{w} + \nu_1 \mathbf{d}_1$$

$$\lambda_2 \tilde{\mathbf{x}}_2 = \boldsymbol{\Omega} \mathbf{w} + \boldsymbol{\tau} + \nu_2 \mathbf{d}_2$$

$$\nu \rightarrow \text{index to the scan line in the image}$$

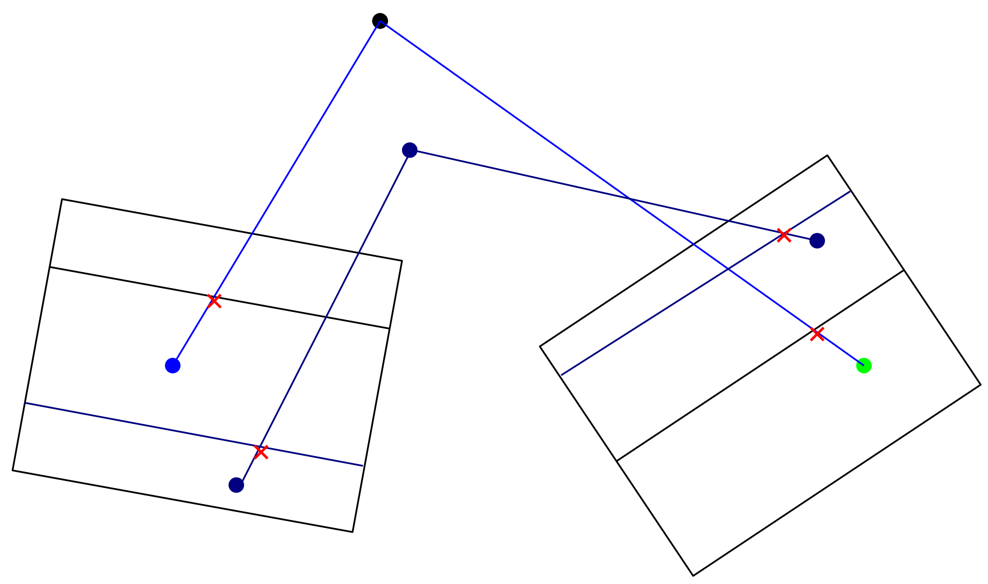
$$\mathbf{d}_i \rightarrow \text{3D velocity for i-}th\text{ viewpoint}$$

Taken from: Y. Dai, H. Li and L. Kneip “Rolling Shutter Camera Relative Pose: Generalized Epipolar Geometry”, arXiv preprint arXiv:1605.00475 (2016).

Epipolar Geometry for Rolling Shutter

- Results in a different essential matrix for every possible combination of ν_1 and ν_2.

$$\mathbf{E}(\nu_1, \nu_2) = (\boldsymbol{\tau} + \nu_2 \mathbf{d}_2 - \nu_1 \boldsymbol{\Omega} \mathbf{d}_1)_\times \boldsymbol{\Omega}$$

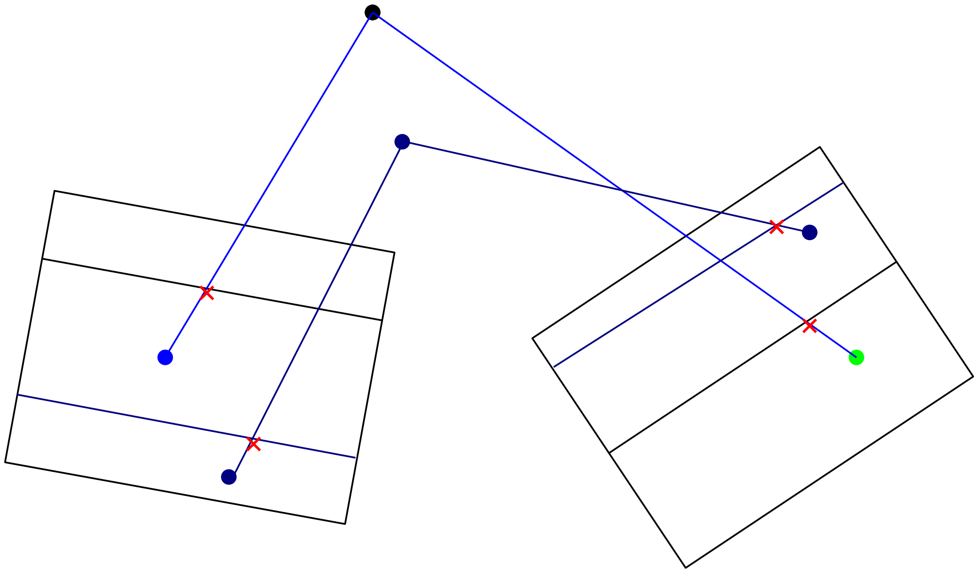


Taken from: Y. Dai, H. Li and L. Kneip "Rolling Shutter Camera Relative Pose: Generalized Epipolar Geometry", arXiv preprint arXiv:1605.00475 (2016).

Epipolar Geometry for Rolling Shutter

- Results in a different essential matrix for every possible combination of ν_1 and ν_2.

$$\mathbf{E}(\nu_1, \nu_2) = (\boldsymbol{\tau} + \nu_2 \mathbf{d}_2 - \nu_1 \boldsymbol{\Omega} \mathbf{d}_1)_\times \boldsymbol{\Omega}$$



How many degrees of freedom?

Taken from: Y. Dai, H. Li and L. Kneip "Rolling Shutter Camera Relative Pose: Generalized Epipolar Geometry", arXiv preprint arXiv:1605.00475 (2016).

Epipolar Geometry for Rolling Shutter

Camera Model	Essential Matrix	Monomials	Degree-of-freedom	Linear Algorithm	Non-linear Algorithm	Motion Parameters
Perspective camera	$\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}$	$(u_i, v_i, 1)$	$3^2 = 9$	8-point	5-point	$\mathbf{R}, \mathbf{t}$
Linear push broom	$\begin{bmatrix} 0 & 0 & f_{13} & f_{14} \\ 0 & 0 & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{bmatrix}$	$(u_i v_i, u_i, v_i, 1)$	$12 = 4^2 - 2^2$	11-point	11-point	$\mathbf{R}, \mathbf{t}, \mathbf{d}_1, \mathbf{d}_2$
Linear rolling shutter	$\begin{bmatrix} 0 & 0 & f_{13} & f_{14} & f_{15} \\ 0 & 0 & f_{23} & f_{24} & f_{25} \\ f_{31} & f_{32} & f_{33} & f_{34} & f_{35} \\ f_{41} & f_{42} & f_{43} & f_{44} & f_{45} \\ f_{51} & f_{52} & f_{53} & f_{54} & f_{55} \end{bmatrix}$	$(u_i^2, u_i v_i, u_i, v_i, 1)$	$21 = 5^2 - 2^2$	20-point	11-point	$\mathbf{R}, \mathbf{t}, \mathbf{d}_1, \mathbf{d}_2$
Uniform push broom	$\begin{bmatrix} 0 & 0 & f_{13} & f_{14} & f_{15} & f_{16} \\ 0 & 0 & f_{23} & f_{24} & f_{25} & f_{26} \\ f_{31} & f_{32} & f_{33} & f_{34} & f_{35} & f_{36} \\ f_{41} & f_{42} & f_{43} & f_{44} & f_{45} & f_{46} \\ f_{51} & f_{52} & f_{53} & f_{54} & f_{55} & f_{56} \\ f_{61} & f_{62} & f_{63} & f_{64} & f_{65} & f_{66} \end{bmatrix}$	$(u_i^2 v_i, u_i^2, u_i v_i, u_i, v_i, 1)$	$32 = 6^2 - 2^2$	31-point	17-point	$\mathbf{R}, \mathbf{t}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{d}_1, \mathbf{d}_2$
Uniform rolling shutter	$\begin{bmatrix} 0 & 0 & f_{13} & f_{14} & f_{15} & f_{16} & f_{17} \\ 0 & 0 & f_{23} & f_{24} & f_{25} & f_{26} & f_{27} \\ f_{31} & f_{32} & f_{33} & f_{34} & f_{35} & f_{36} & f_{37} \\ f_{41} & f_{42} & f_{43} & f_{44} & f_{45} & f_{46} & f_{47} \\ f_{51} & f_{52} & f_{53} & f_{54} & f_{55} & f_{56} & f_{57} \\ f_{61} & f_{62} & f_{63} & f_{64} & f_{65} & f_{66} & f_{67} \\ f_{71} & f_{72} & f_{73} & f_{74} & f_{75} & f_{76} & f_{77} \end{bmatrix}$	$(u_i^3, u_i^2 v_i, u_i^2, u_i v_i, u_i, v_i, 1)$	$45 = 7^2 - 2^2$	44-point	17-point	$\mathbf{R}, \mathbf{t}, \mathbf{w}_1, \mathbf{w}_2, \mathbf{d}_1, \mathbf{d}_2$

Taken from: Y. Dai, H. Li and L. Kneip "Rolling Shutter Camera Relative Pose: Generalized Epipolar Geometry", arXiv preprint arXiv:1605.00475 (2016).

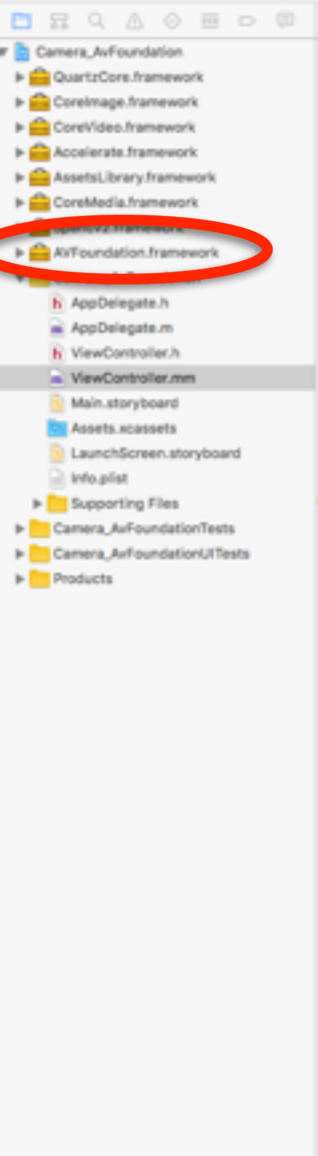
Accessing the Camera in iOS

Camera_AvFoundation > Camera_AvFoundation > ViewController.mm > @implementation ViewController

- Camera_AvFoundation
  - QuartzCore.framework
  - CoreImage.framework
  - CoreVideo.framework
  - Accelerate.framework
  - AssetsLibrary.framework
  - CoreMedia.framework
  - opencv2.framework
  - AVFoundation.framework
  - Camera_AvFoundation
    - AppDelegate.h
    - AppDelegate.m
    - ViewController.h
    - ViewController.mm
    - Main.storyboard
    - Assets.xcassets
    - LaunchScreen.storyboard
    - Info.plist
    - Supporting Files
  - Camera_AvFoundationTests
  - Camera_AvFoundationUITests
  - Products

```objc
//
//  ViewController.m
//  Camera_AvFoundation
//
//  Created by Simon Lucey on 9/7/16.
//  Copyright © 2016 CMU_16623. All rights reserved.
//

#import "ViewController.h"
#include <iostream>

@interface ViewController();
@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    // Initialize the view
    imageView_ = [[UIImageView alloc] initWithFrame:CGRectMake(0.0, 0.0, self.view.frame.size.
        width, self.view.frame.size.height)];
    [self.view addSubview:imageView_];

    // Initialize the video camera
    self.videoCamera = [[CvVideoCamera alloc] initWithParentView:imageView_];
    self.videoCamera.delegate = self;
    self.videoCamera.defaultAVCaptureDevicePosition = AVCaptureDevicePositionFront;
    self.videoCamera.defaultAVCaptureSessionPreset = AVCaptureSessionPreset640x480;
    self.videoCamera.defaultAVCaptureVideoOrientation = AVCaptureVideoOrientationPortrait;
    self.videoCamera.defaultFPS = 30;

    // Finally show the output
    [videoCamera start];
    isCapturing = YES;
```

Accessing the Camera in iOS



Camera_AvFoundation › Camera_AvFoundation › ViewController.mm › @implementation ViewController

```
//
//  ViewController.m
//  Camera_AvFoundation
//
//  Created by Simon Lucey on 9/7/16.
//  Copyright © 2016 CMU_16623. All rights reserved.
//

#import "ViewController.h"
#include <iostream>

@interface ViewController();
@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    // Initialize the view
    imageView_ = [[UIImageView alloc] initWithFrame:CGRectMake(0.0, 0.0, self.view.frame.size.
        width, self.view.frame.size.height)];
    [self.view addSubview:imageView_];

    // Initialize the video camera
    self.videoCamera = [[CvVideoCamera alloc] initWithParentView:imageView_];
    self.videoCamera.delegate = self;
    self.videoCamera.defaultAVCaptureDevicePosition = AVCaptureDevicePositionFront;
    self.videoCamera.defaultAVCaptureSessionPreset = AVCaptureSessionPreset640x480;
    self.videoCamera.defaultAVCaptureVideoOrientation = AVCaptureVideoOrientationPortrait;
    self.videoCamera.defaultFPS = 30;

    // Finally show the output
    [videoCamera start];
    isCapturing = YES;
```

Accessing the Camera in iOS

Camera_AvFoundation 〉 Camera_AvFoundation 〉 ViewController.mm 〉 @implementation ViewController

- Camera_AvFoundation
  - QuartzCore.framework
  - CoreImage.framework
  - CoreVideo.framework
  - Accelerate.framework
  - AssetsLibrary.framework
  - CoreMedia.framework
  - AVFoundation.framework
    - AppDelegate.h
    - AppDelegate.m
    - ViewController.h
    - ViewController.mm
    - Main.storyboard
    - Assets.xcassets
    - LaunchScreen.storyboard
    - Info.plist
    - Supporting Files
  - Camera_AvFoundationTests
  - Camera_AvFoundationUITests
  - Products

```
//
//  ViewController.m
//  Camera_AvFoundation
//
//  Created by Simon Lucey on 9/7/16.
//  Copyright © 2016 CMU_16623. All rights reserved.
//

#import "ViewController.h"
#include <iostream>

@interface ViewController();
@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    // Initialize the view
    imageView_ = [[UIImageView alloc] initWithFrame:CGRectMake(0.0, 0.0, self.view.frame.size.
        width, self.view.frame.size.height)];
    [self.view addSubview:imageView_];

    // Initialize the video camera
    self.videoCamera = [[CvVideoCamera alloc] initWithParentView:imageView_];
    self.videoCamera.delegate = self;
    self.videoCamera.defaultAVCaptureDevicePosition = AVCaptureDevicePositionFront;
    self.videoCamera.defaultAVCaptureSessionPreset = AVCaptureSessionPreset640x480;
    self.videoCamera.defaultAVCaptureVideoOrientation = AVCaptureVideoOrientationPortrait;
    self.videoCamera.defaultFPS = 30;

    // Finally show the output
    [videoCamera start];
    isCapturing = YES;
```

Today

- CCD vs CMOS cameras.
- Rolling Shutter Epipolar Geometry
- Inertial Measurement Units (IMU)

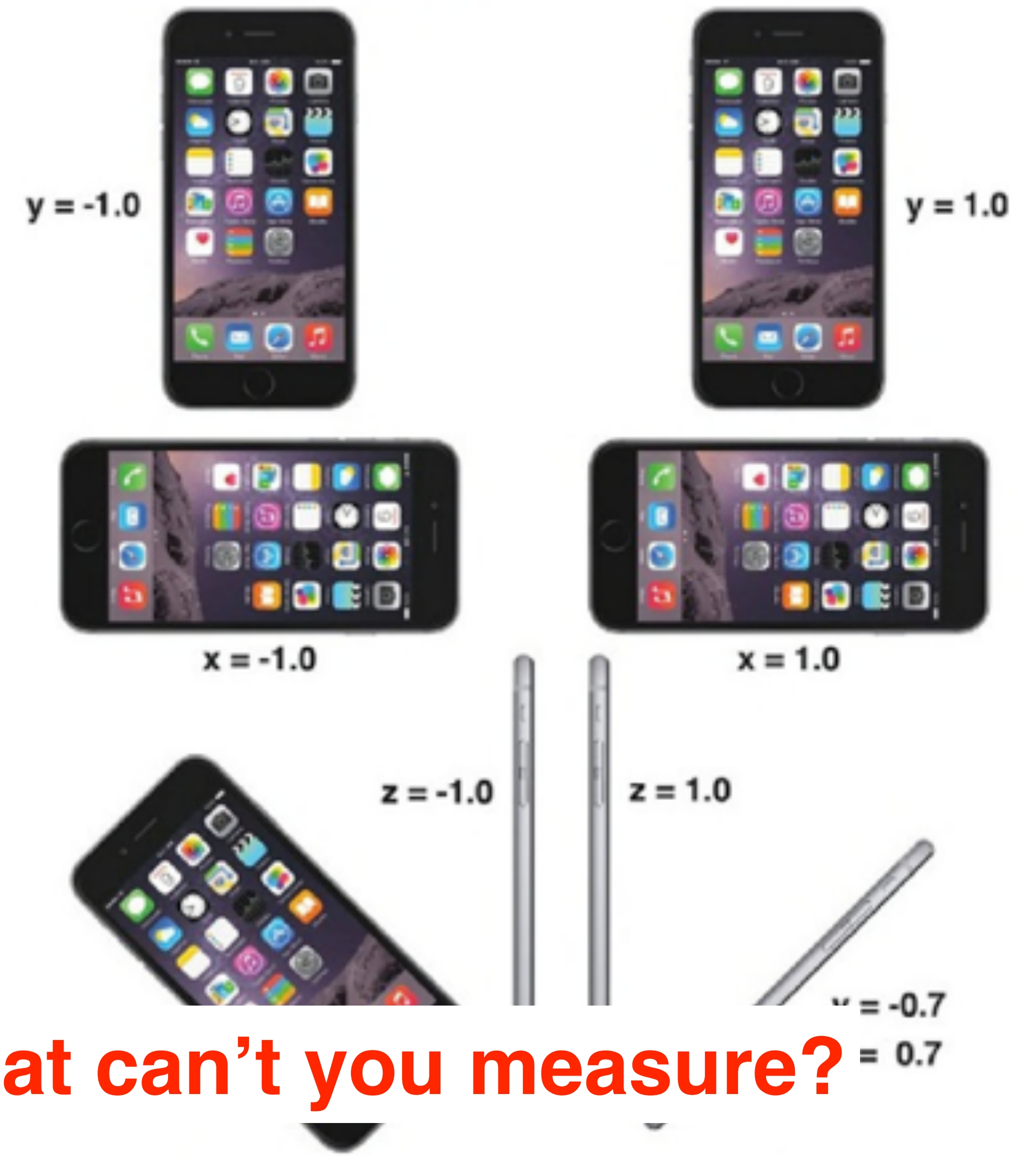
Inertial Measurement Unit

- Measures a device's specific force, angular rate & magnetic field.
- Composed of,
    - Accelerometer.
    - Gyroscope.
    - Magnetometer.
- Historically used heavily within navigation and robotic systems.
- More recently have become common place in smart devices.

Accelerometer

Accelerometer

Accelerometer



What can’t you measure?

Gyroscope

IMU Example in iOS

- Good example of using IMU in iOS can be found at,

https://github.com/nscookbook/recipe19

- Or better yet, if you have git installed you can type from the command line.

```
$ git clone https://github.com/NSCookbook/recipe19.git
```

- Good tutorial about how code works can be found at,

http://nscookbook.com/2013/03/ios-programming-recipe-19-using-core-motion-to-access-gyro-and-accelerometer/

Accessing the IMU in iOS

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    currentMaxAccelX = 0;
    currentMaxAccelY = 0;
    currentMaxAccelZ = 0;

    currentMaxRotX = 0;
    currentMaxRotY = 0;
    currentMaxRotZ = 0;

    self.motionManager = [[CMMotionManager alloc] init];
    self.motionManager.accelerometerUpdateInterval = .2;
    self.motionManager.gyroUpdateInterval = .2;

    [self.motionManager startAccelerometerUpdatesToQueue:[NSOperationQueue currentQueue]
                                             withHandler:^(CMAccelerometerData *accelerometerData, NSError *
                                                 error) {
        [self outputAccelertionData:accelerometerData.acceleration];
                                                 if(error){

                                                     NSLog(@"%@", error);
                                                 }
    }];

    [self.motionManager startGyroUpdatesToQueue:[NSOperationQueue currentQueue]
                                    withHandler:^(CMGyroData *gyroData, NSError *error) {
                                        [self outputRotationData:gyroData.rotationRate];
                                    }];
```

Accessing the IMU in iOS

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    currentMaxAccelX = 0;
    currentMaxAccelY = 0;
    currentMaxAccelZ = 0;

    currentMaxRotX = 0;
    currentMaxRotY = 0;
    currentMaxRotZ = 0;

    self.motionManager = [[CMMotionManager alloc] init];
    self.motionManager.accelerometerUpdateInterval = .2;
    self.motionManager.gyroUpdateInterval = .2;

    [self.motionManager startAccelerometerUpdatesToQueue:[NSOperationQueue currentQueue]
                                             withHandler:^(CMAccelerometerData *accelerometerData, NSError *
                                                 error) {
        [self outputAccelertionData:accelerometerData.acceleration];
                                                 if(error){

                                                     NSLog(@"%@", error);
                                                 }
    }];

    [self.motionManager startGyroUpdatesToQueue:[NSOperationQueue currentQueue]
                                    withHandler:^(CMGyroData *gyroData, NSError *error) {
                                        [self outputRotationData:gyroData.rotationRate];
                                    }];
```

Accessing the IMU in iOS

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    currentMaxAccelX = 0;
    currentMaxAccelY = 0;
    currentMaxAccelZ = 0;

    currentMaxRotX = 0;
    currentMaxRotY = 0;
    currentMaxRotZ = 0;

    self.motionManager = [[CMMotionManager alloc] init];
    self.motionManager.accelerometerUpdateInterval = .2;
    self.motionManager.gyroUpdateInterval = .2;

    [self.motionManager startAccelerometerUpdatesToQueue:[NSOperationQueue currentQueue]
                                             withHandler:^(CMAccelerometerData *accelerometerData, NSError *
                                                 error) {
        [self outputAccelertionData:accelerometerData.acceleration];
                                                 if(error){

                                                     NSLog(@"%@", error);
                                                 }
    }];

    [self.motionManager startGyroUpdatesToQueue:[NSOperationQueue currentQueue]
                                    withHandler:^(CMGyroData *gyroData, NSError *error) {
                                        [self outputRotationData:gyroData.rotationRate];
                                    }];
```

Accessing the IMU in iOS

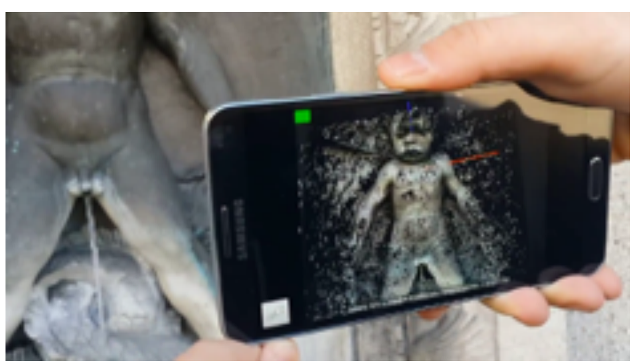
```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    currentMaxAccelX = 0;
    currentMaxAccelY = 0;
    currentMaxAccelZ = 0;

    currentMaxRotX = 0;
    currentMaxRotY = 0;
    currentMaxRotZ = 0;

    self.motionManager = [[CMMotionManager alloc] init];
    self.motionManager.accelerometerUpdateInterval = .2;
    self.motionManager.gyroUpdateInterval = .2;

    [self.motionManager startAccelerometerUpdatesToQueue:[NSOperationQueue currentQueue]
                                             withHandler:^(CMAccelerometerData *accelerometerData, NSError *
                                                 error) {
        [self outputAccelertionData:accelerometerData.acceleration];
                                                 if(error){

                                                     NSLog(@"%@", error);
                                                 }
    }];

    [self.motionManager startGyroUpdatesToQueue:[NSOperationQueue currentQueue]
                                    withHandler:^(CMGyroData *gyroData, NSError *error) {
                                        [self outputRotationData:gyroData.rotationRate];
                                    }];
```

Robotics - Monocular Camera + IMU

- Jones, E., Vedaldi, A., Soatto, S.: Inertial structure from motion with autocalibration. In: Workshop on Dynamical Vision. (2007)
- Weiss, S., Achtelik, M.W., Lynen, S., Achtelik, M.C., Kneip, L., Chli, M., Siegwart, R.: Monocular vision for long-term micro aerial vehicle state estimation: A compendium. Journal of Field Robotics 30(5) (2013) 803–831
- Nutzi, G., Weiss, S., Scaramuzza, D., Siegwart, R.: Fusion of IMU and vision for absolute scale estimation in monocular slam. Journal of Intelligent & Robotic Systems 61(1-4) (2011) 287–299
- Li, M., Kim, B.H., Mourikis, A.I.: Real-time motion tracking on a cellphone using inertial sensing and a rolling-shutter camera. In: IEEE International Conference on Robotics and Automation (ICRA). (2013) 4712–4719

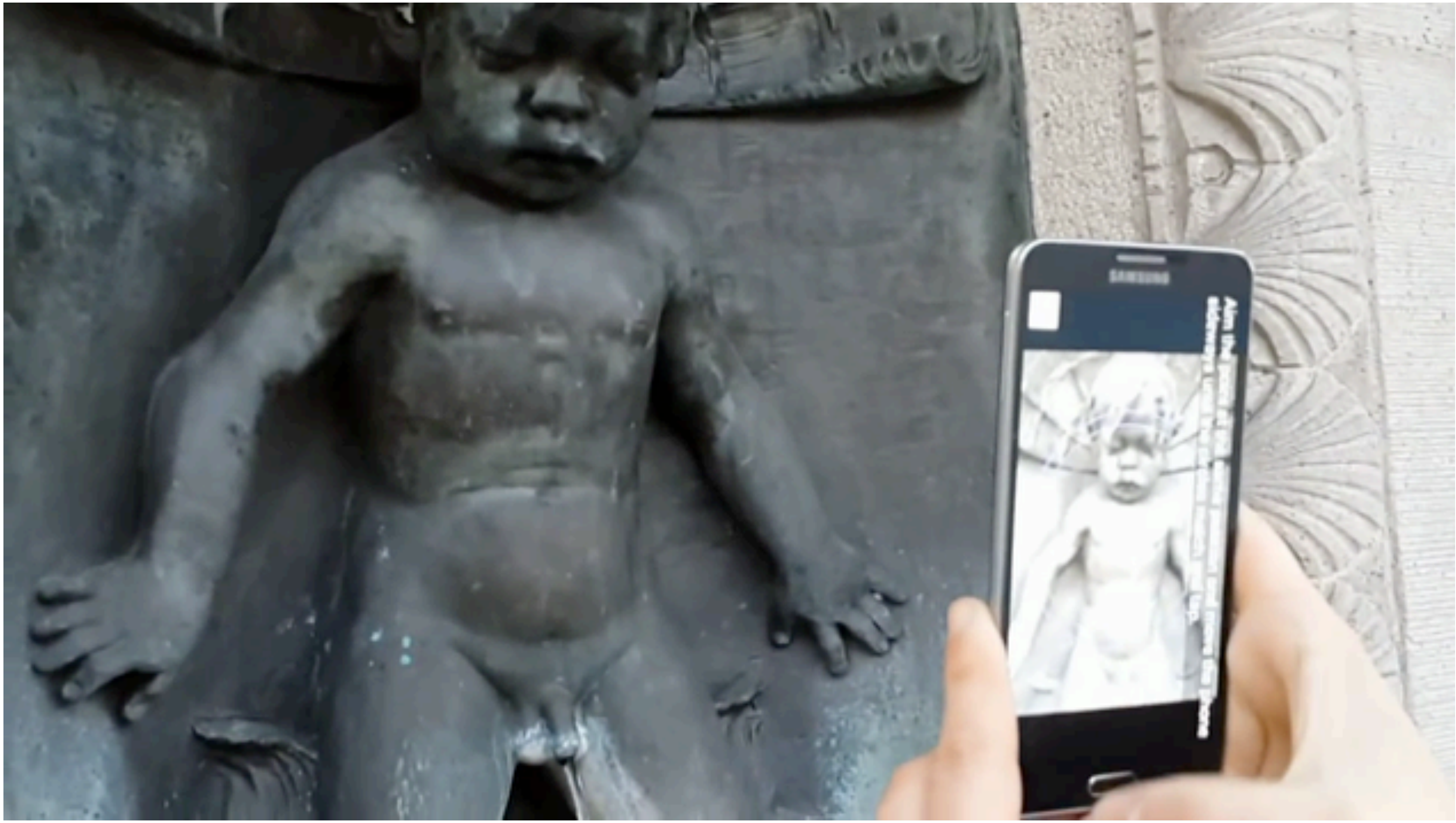
Mobile Solutions

- Tanskanen et al. - ETH Zurich
- Generates accurate point-cloud using SLAM (PTAM)
- Integrates IMU for scale



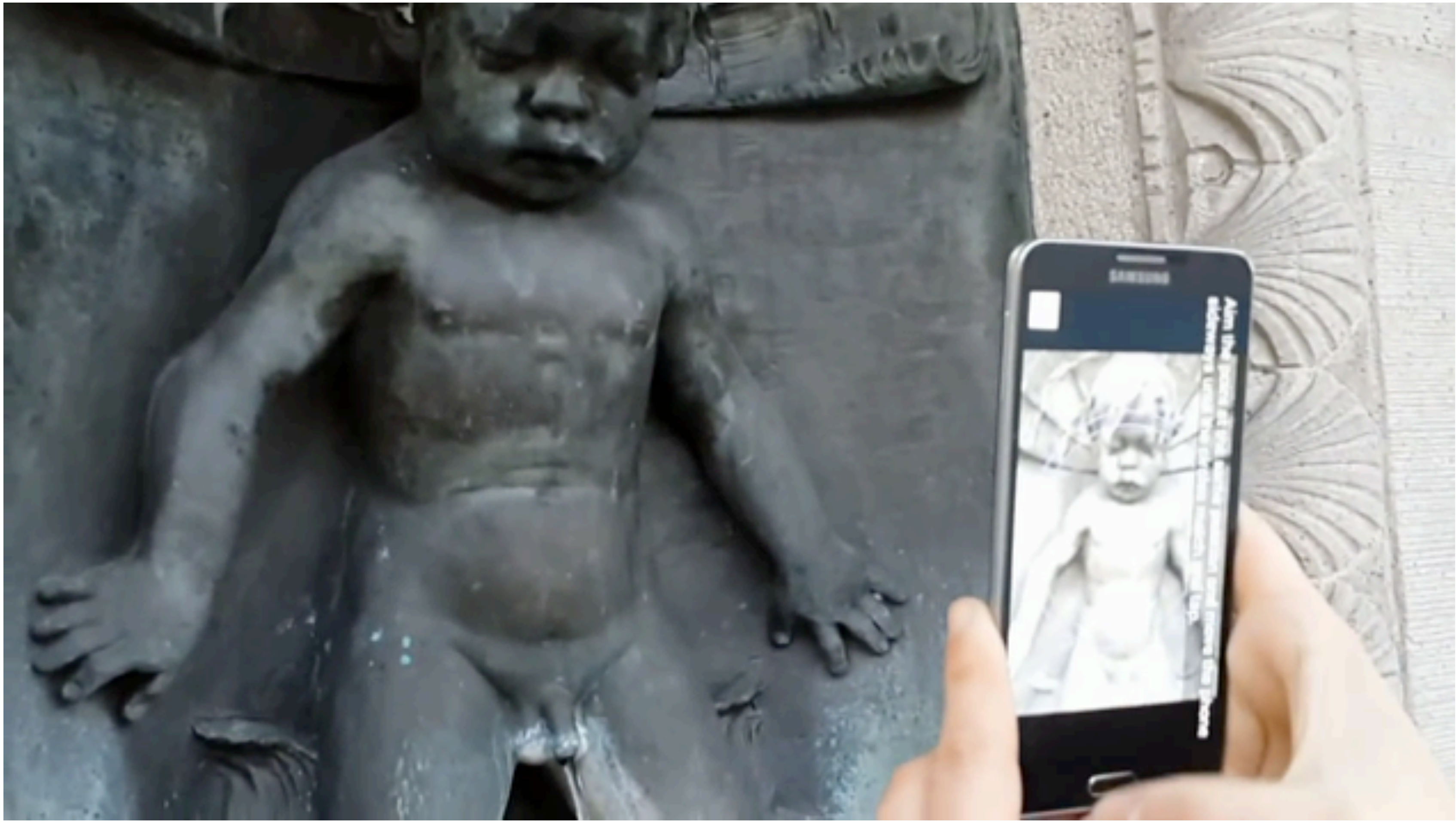
P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, M. Pollefeys : Live metric 3d reconstruction on mobile phones. (ICCV 2013)

Mobile Visual SLAM + IMU



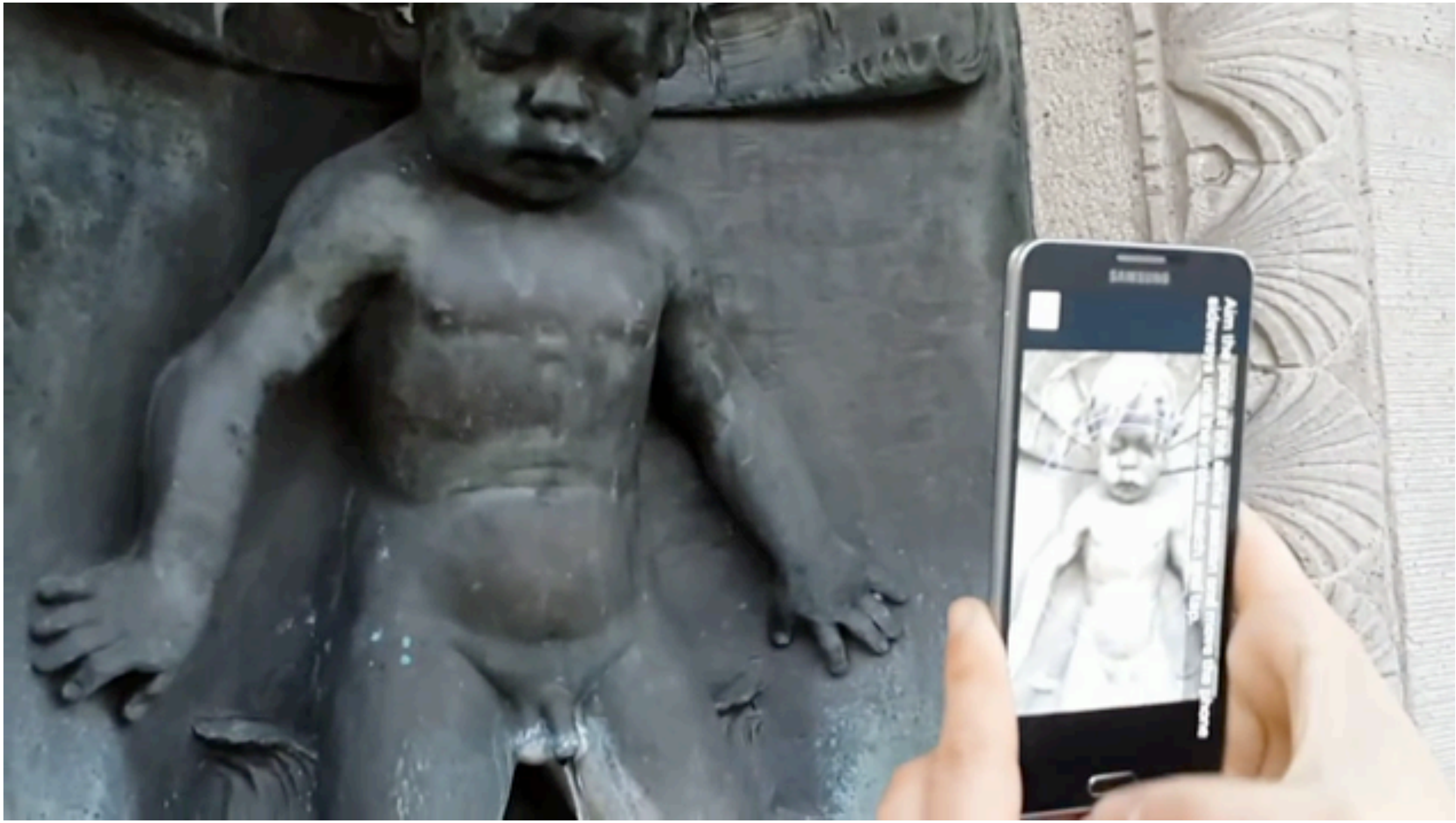
P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, M. Pollefeys : Live metric 3d reconstruction on mobile phones. (ICCV 2013)

Mobile Visual SLAM + IMU

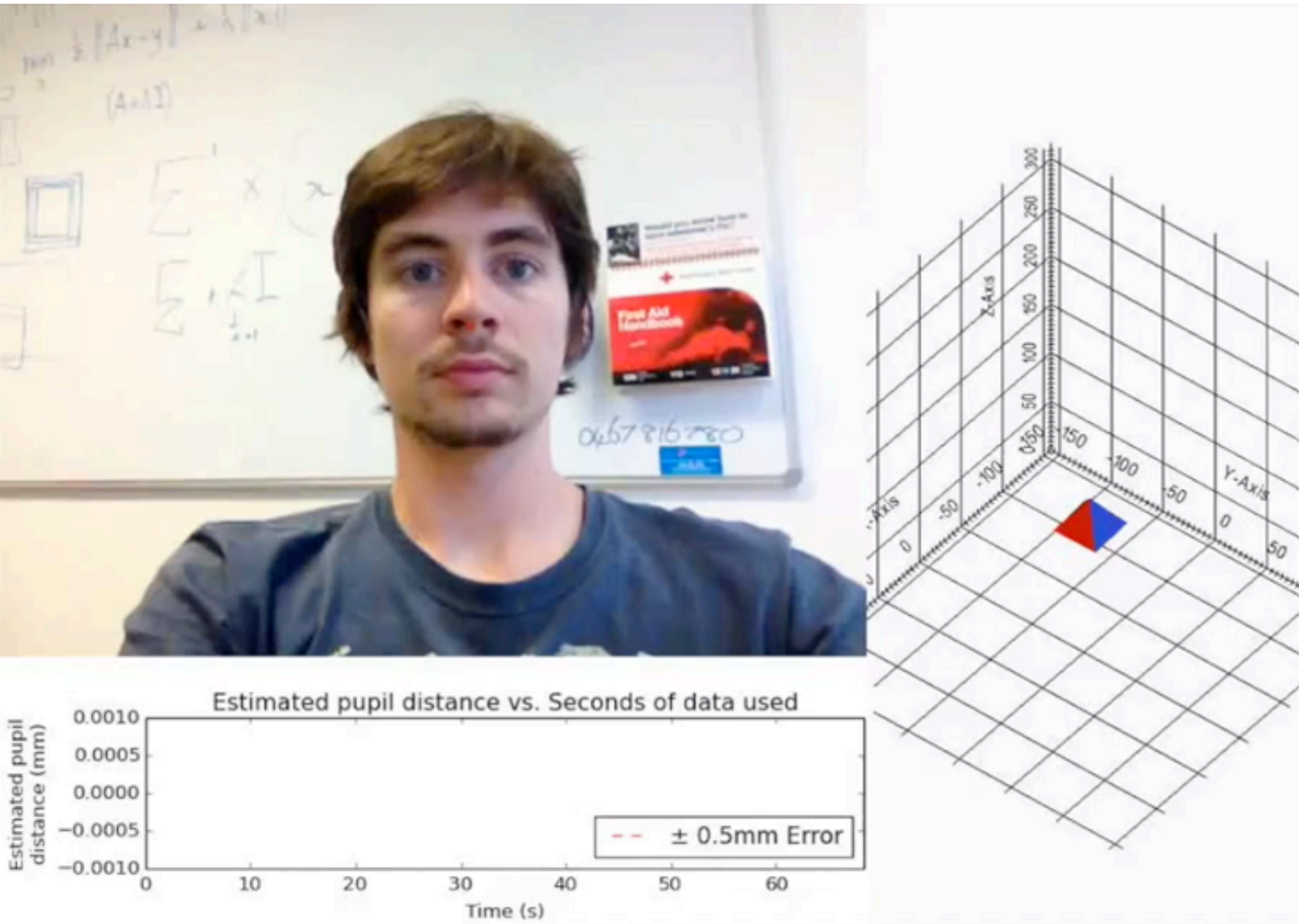


P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, M. Pollefeys : Live metric 3d reconstruction on mobile phones. (ICCV 2013)

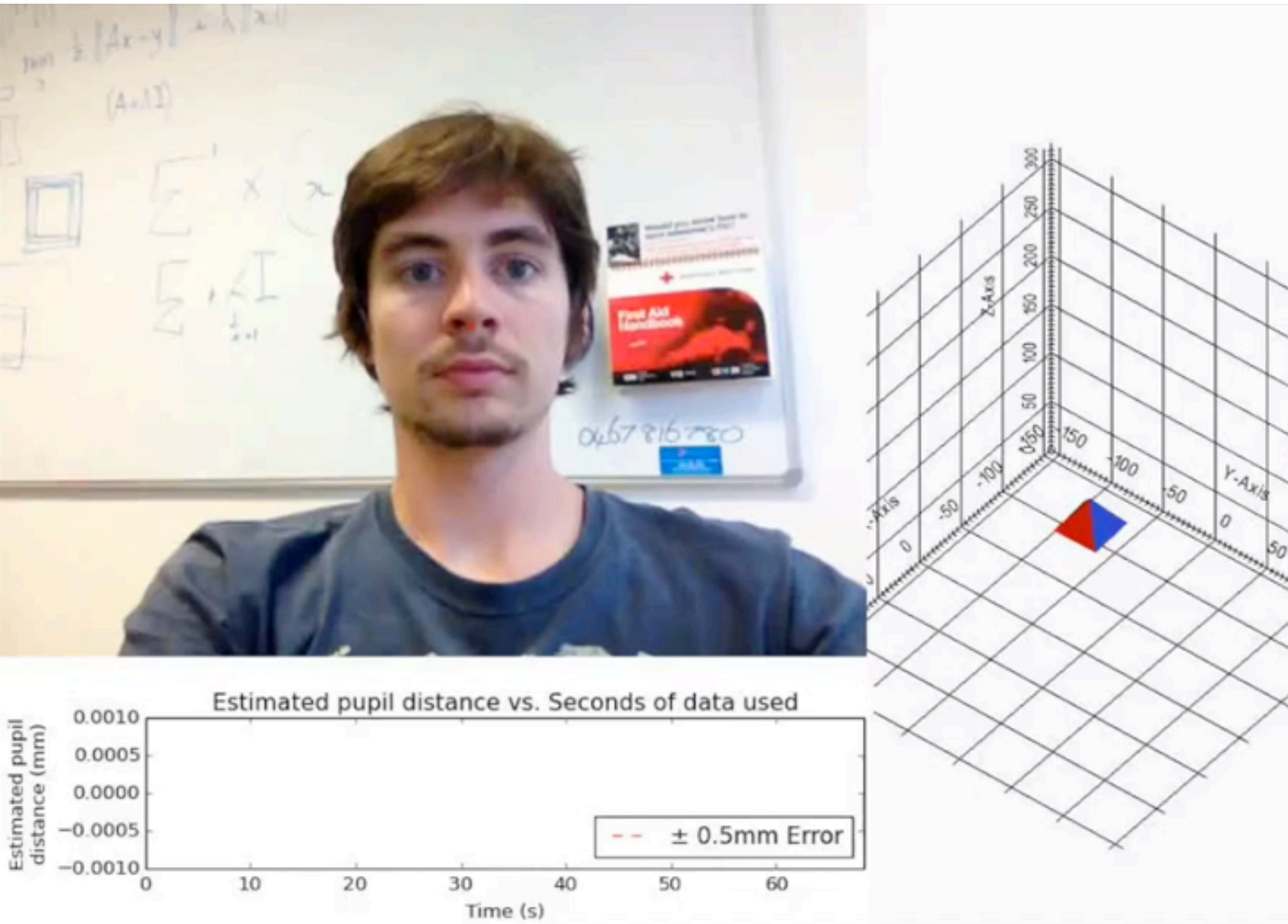
Mobile Visual SLAM + IMU



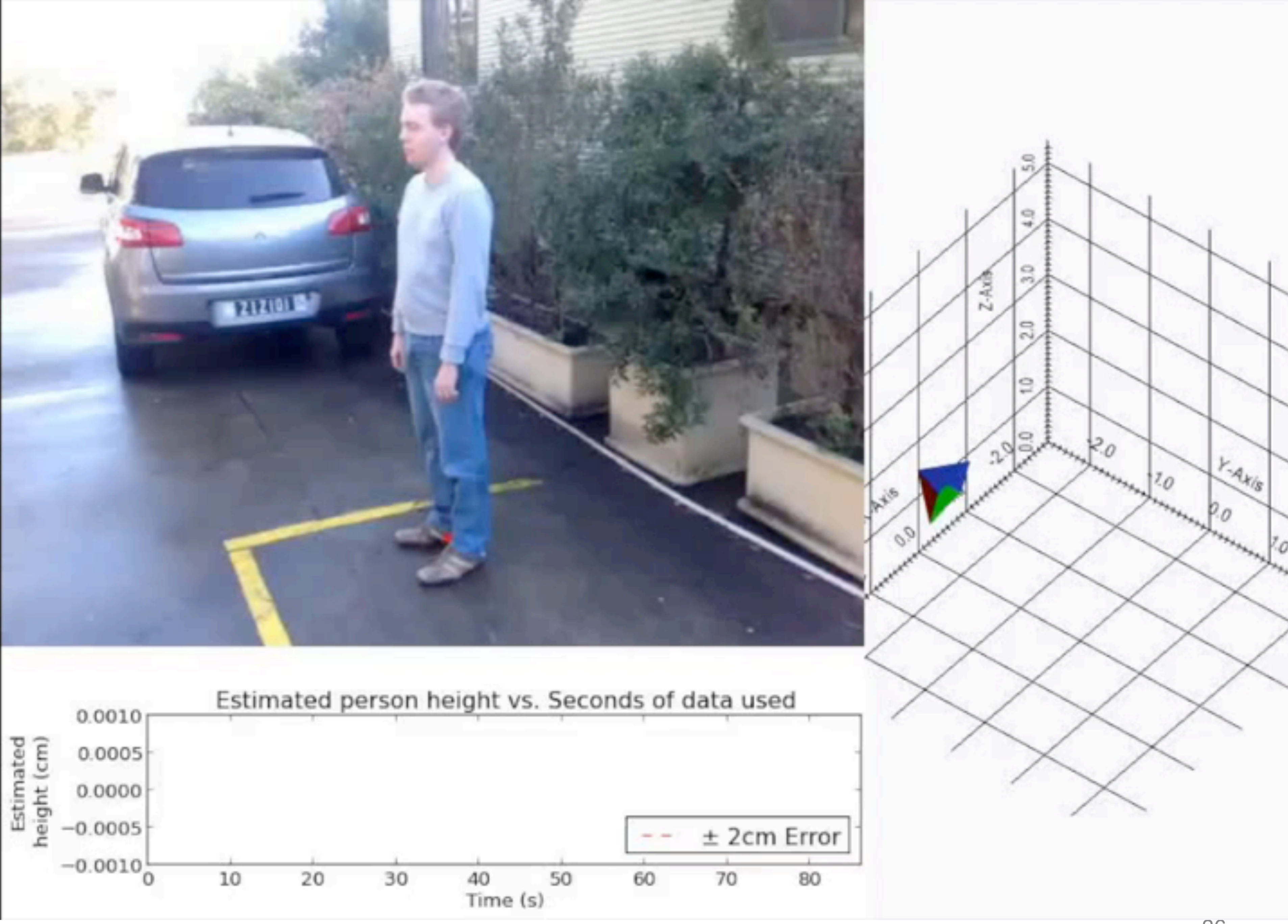
P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, M. Pollefeys : Live metric 3d reconstruction on mobile phones. (ICCV 2013)



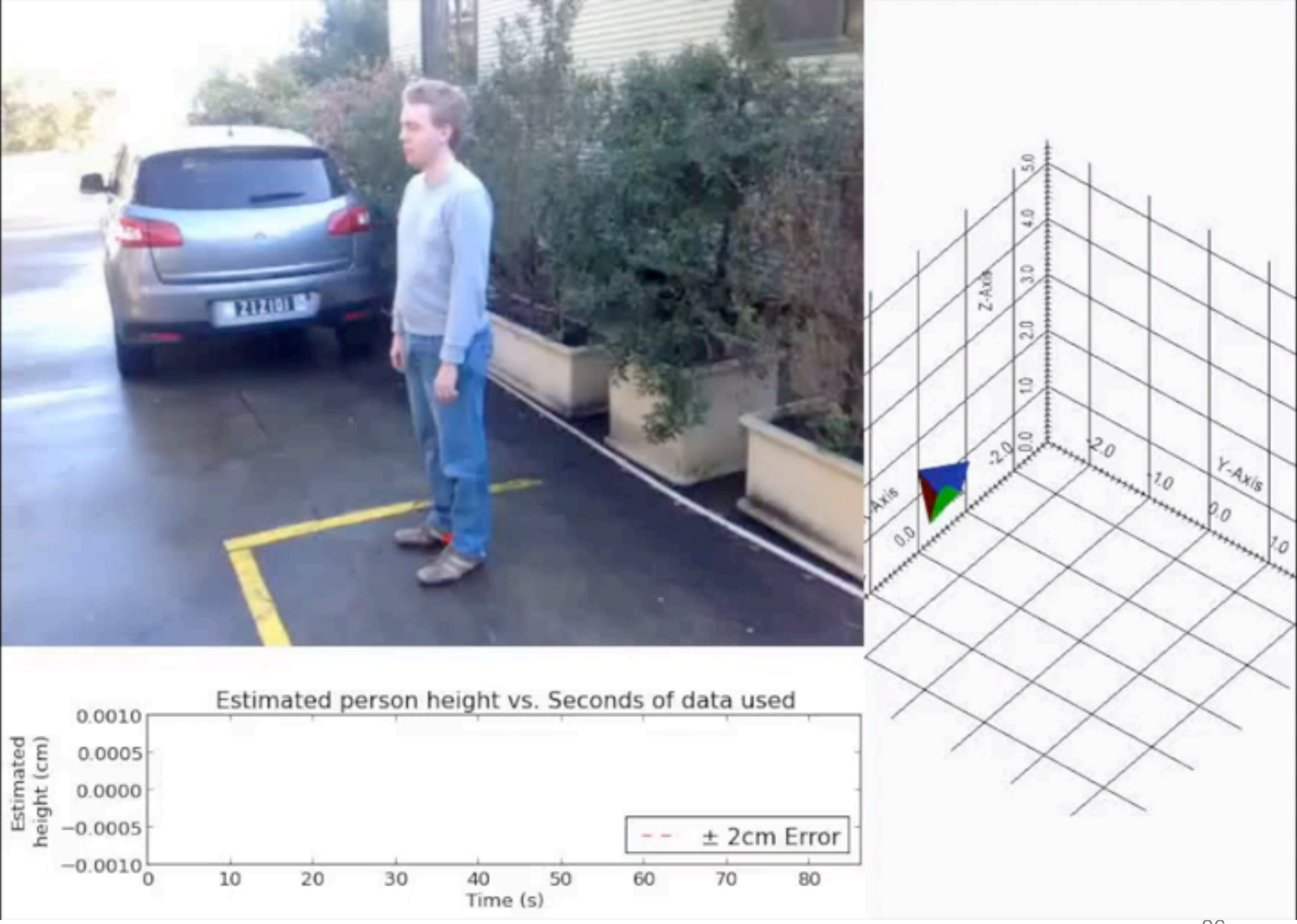
C. Ham, S. Singh, and S. Lucey: Handwaving away scale. (ECCV 2014)



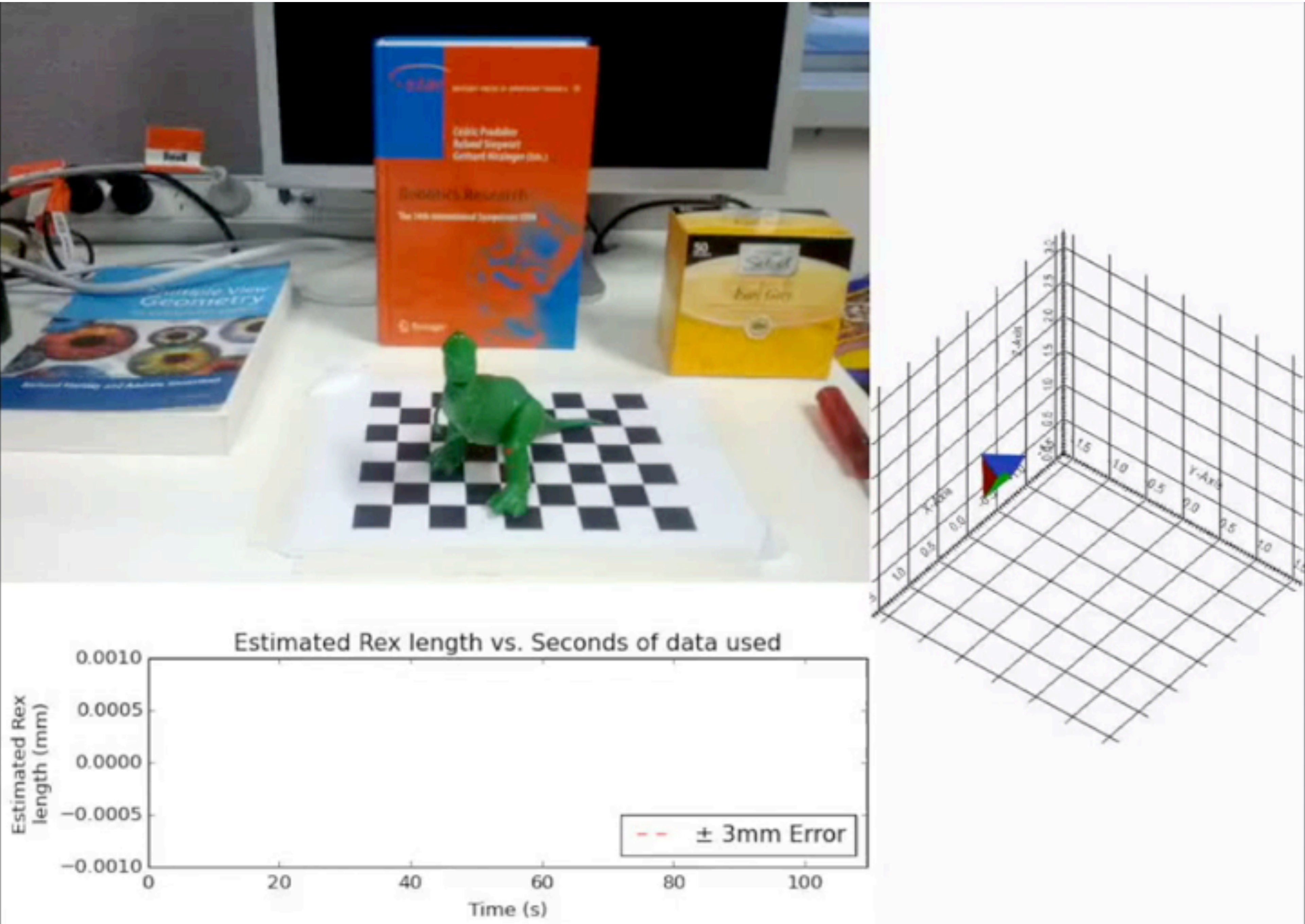
C. Ham, S. Singh, and S. Lucey: Handwaving away scale. (ECCV 2014)



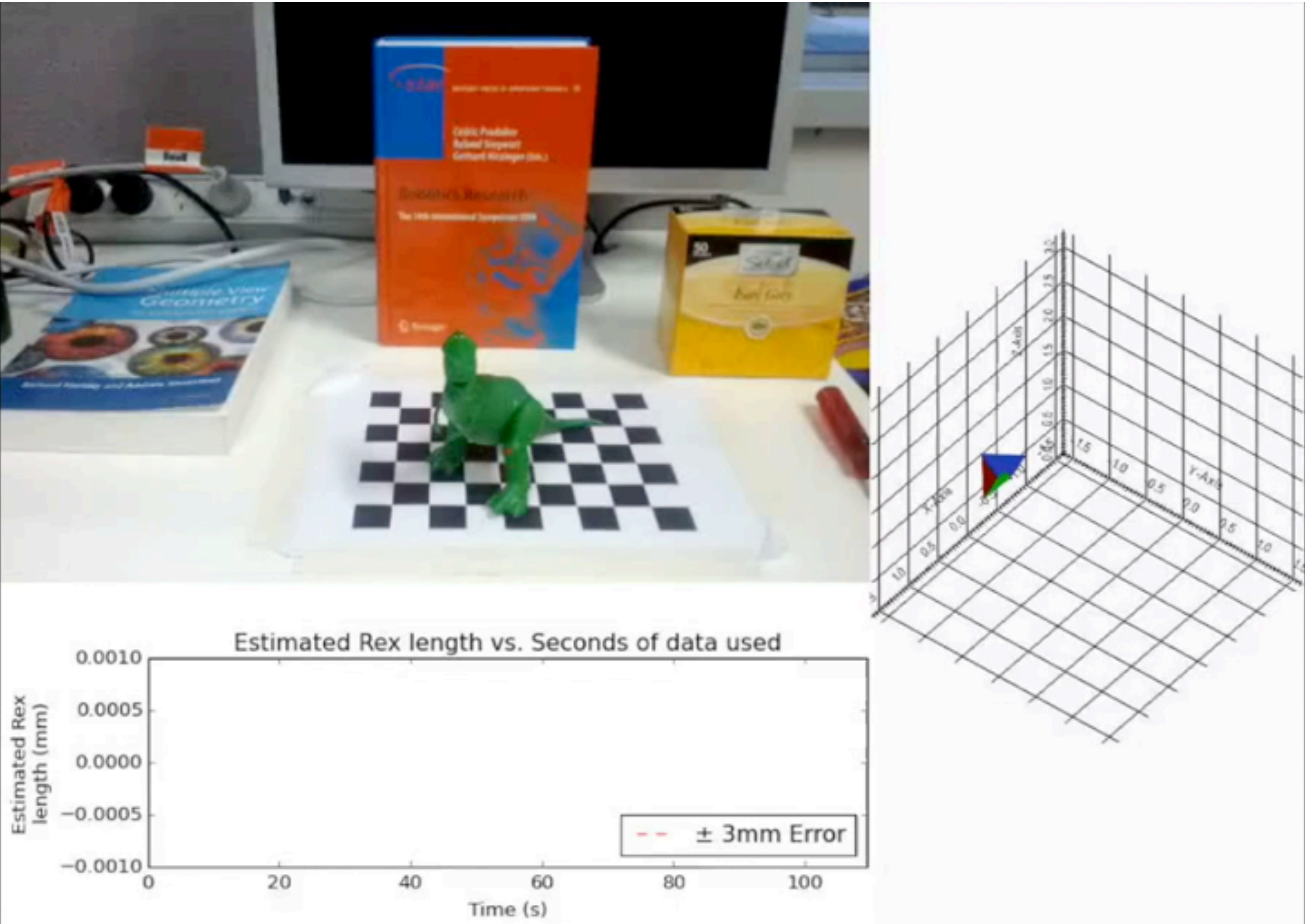
C. Ham, S. Singh, and S. Lucey: Handwaving away scale. (ECCV 2014)



C. Ham, S. Singh, and S. Lucey: Handwaving away scale. (ECCV 2014)



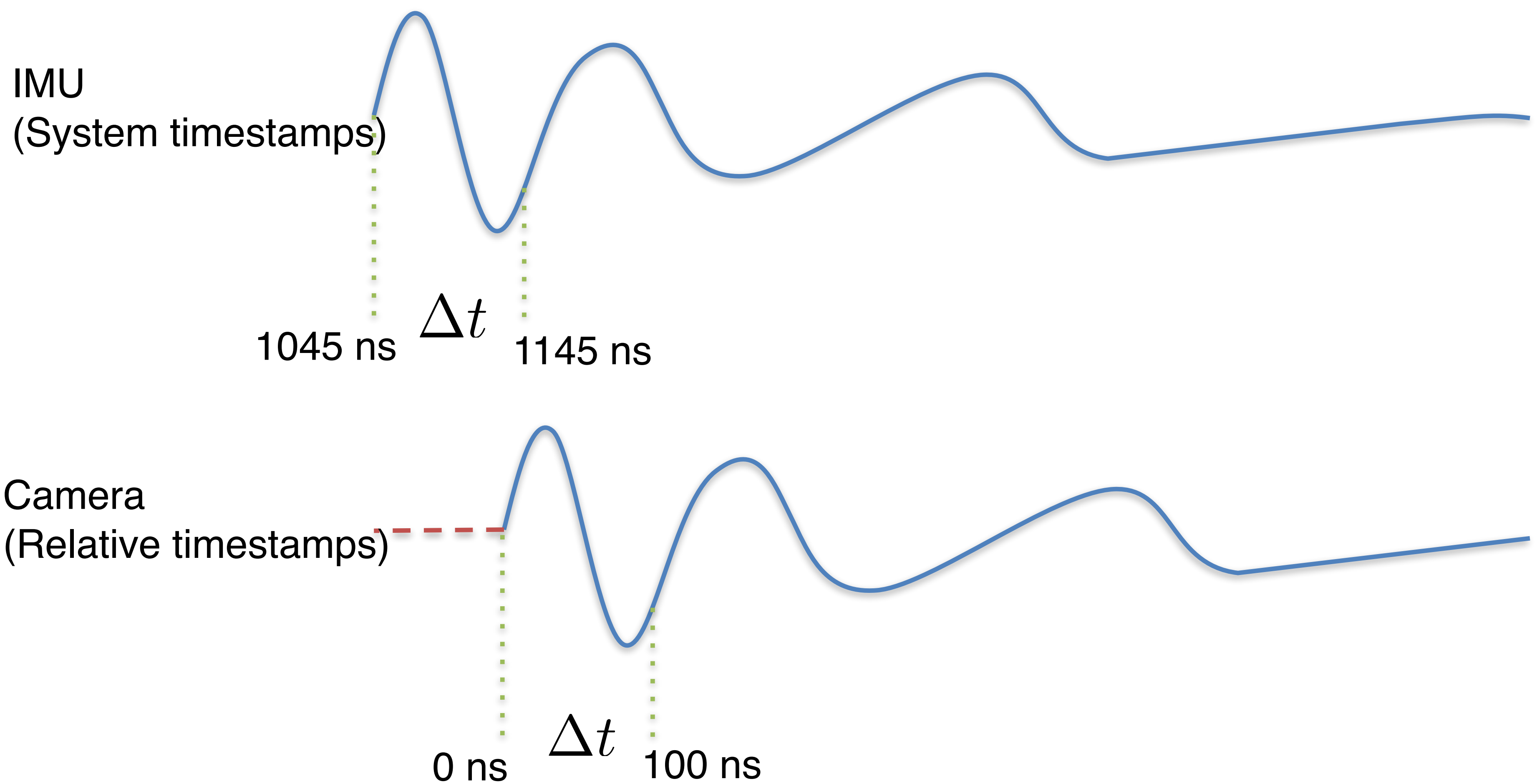
C. Ham, S. Singh, and S. Lucey: Handwaving away scale. (ECCV 2014)



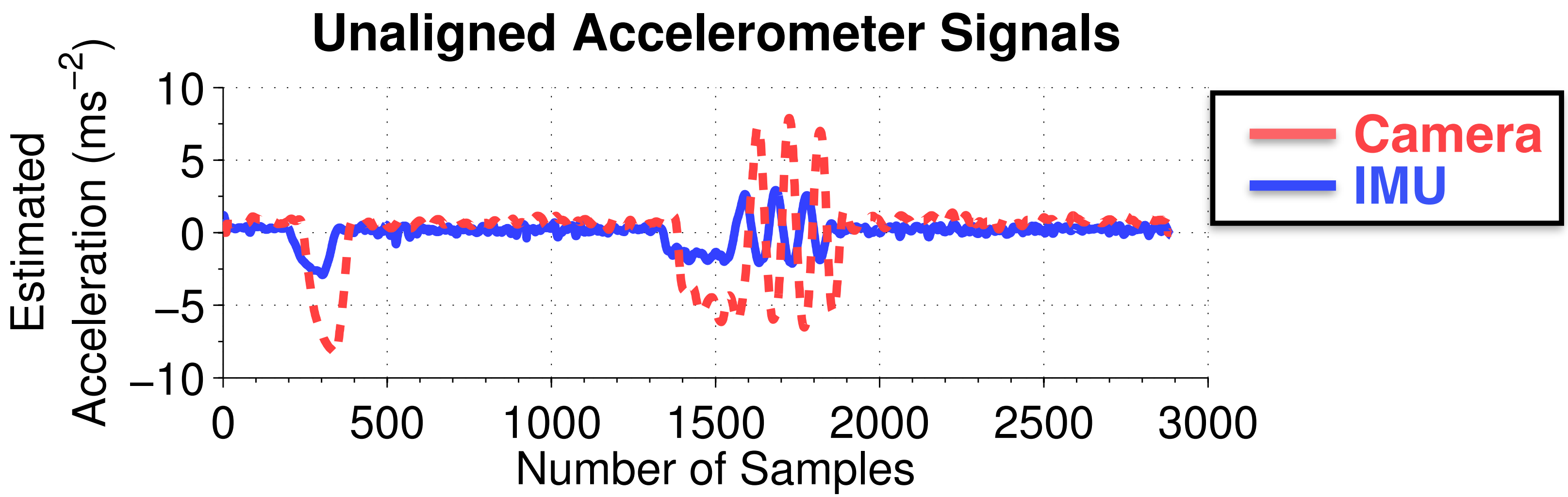
C. Ham, S. Singh, and S. Lucey: Handwaving away scale. (ECCV 2014)

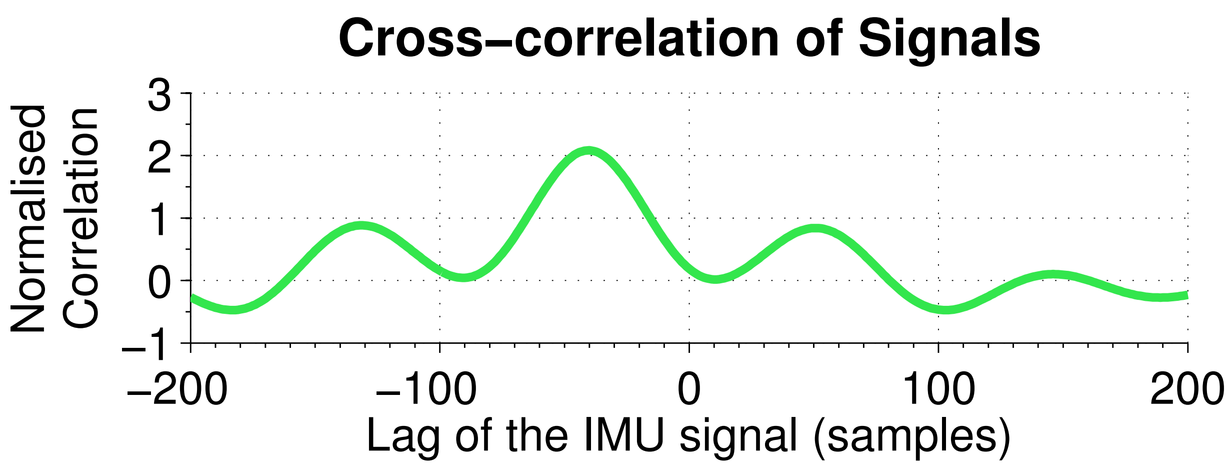
Mobile Platform Issues

- IMU and Camera time stamped differently



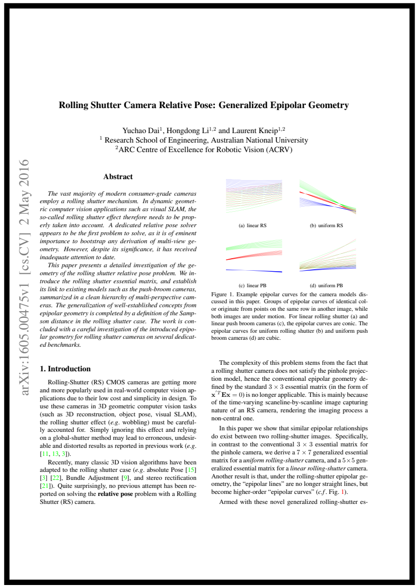
Auto-Correlation





C. Ham, S. Singh, and S. Lucey: Handwaving away scale. (ECCV 2014)

More to read…



arXiv:1605.00475v1 [cs.CV] 2 May 2016

Rolling Shutter Camera Relative Pose: Generalized Epipolar Geometry

Yuchao Dai[1], Hongdong Li[1,2] and Laurent Kneip[1,2]
[1] Research School of Engineering, Australian National University
[2]ARC Centre of Excellence for Robotic Vision (ACRV)

Abstract

The vast majority of modern consumer-grade cameras employ a rolling shutter mechanism. In dynamic geometric computer vision applications such as visual SLAM, the so-called rolling shutter effect therefore needs to be properly taken into account. A dedicated relative pose solver appears to be the first problem to solve, as it is of eminent importance to bootstrap any derivation of multi-view geometry. However, despite its significance, it has received inadequate attention to date.

This paper presents a detailed investigation of the geometry of the rolling shutter relative pose problem. We introduce the rolling shutter essential matrix, and establish its link to existing models such as the push-broom cameras, summarized in a clean hierarchy of multi-perspective cameras. The generalization of well-established concepts from epipolar geometry is completed by a definition of the Sampson distance in the rolling shutter case. The work is concluded with a careful investigation of the introduced epipolar geometry for rolling shutter cameras on several dedicated benchmarks.

(a) linear RS (b) uniform RS

(c) linear PB (d) uniform PB

Figure 1. Example epipolar curves for the camera models discussed in this paper. Groups of epipolar curves of identical color originate from points on the same row in another image, while both images are under motion. For linear rolling shutter (a) and linear push broom cameras (c), the epipolar curves are conic. The epipolar curves for uniform rolling shutter (b) and uniform push broom cameras (d) are cubic.

1. Introduction

Rolling-Shutter (RS) CMOS cameras are getting more and more popularly used in real-world computer vision applications due to their low cost and simplicity in design. To use these cameras in 3D geometric computer vision tasks (such as 3D reconstruction, object pose, visual SLAM), the rolling shutter effect (*e.g*. wobbling) must be carefully accounted for. Simply ignoring this effect and relying on a global-shutter method may lead to erroneous, undesirable and distorted results as reported in previous work (*e.g*. [11, 13, 3]).

Recently, many classic 3D vision algorithms have been adapted to the rolling shutter case (*e.g*. absolute Pose [15] [3] [22], Bundle Adjustment [9], and stereo rectification [21]). Quite surprisingly, no previous attempt has been reported on solving the **relative pose** problem with a Rolling Shutter (RS) camera.

The complexity of this problem stems from the fact that a rolling shutter camera does not satisfy the pinhole projection model, hence the conventional epipolar geometry defined by the standard 3×3 essential matrix (in the form of $\mathbf{x}'^T\mathbf{E}\mathbf{x} = 0$) is no longer applicable. This is mainly because of the time-varying scaneline-by-scanline image capturing nature of an RS camera, rendering the imaging process a non-central one.

In this paper we show that similar epipolar relationships do exist between two rolling-shutter images. Specifically, in contrast to the conventional 3×3 essential matrix for the pinhole camera, we derive a 7×7 generalized essential matrix for a *uniform rolling-shutter* camera, and a 5×5 generalized essential matrix for a *linear rolling-shutter* camera. Another result is that, under the rolling-shutter epipolar geometry, the "epipolar lines" are no longer straight lines, but become higher-order "epipolar curves" (*c.f*. Fig. 1).

Armed with these novel generalized rolling-shutter es-

- Y. Dai, H. Li and L. Kneip "Rolling Shutter Camera Relative Pose: Generalized Epipolar Geometry", arXiv preprint arXiv:1605.00475 (2016).