

WiSE-MNet (Wireless Multimedia Sensor Networks)

Overview & hands-on

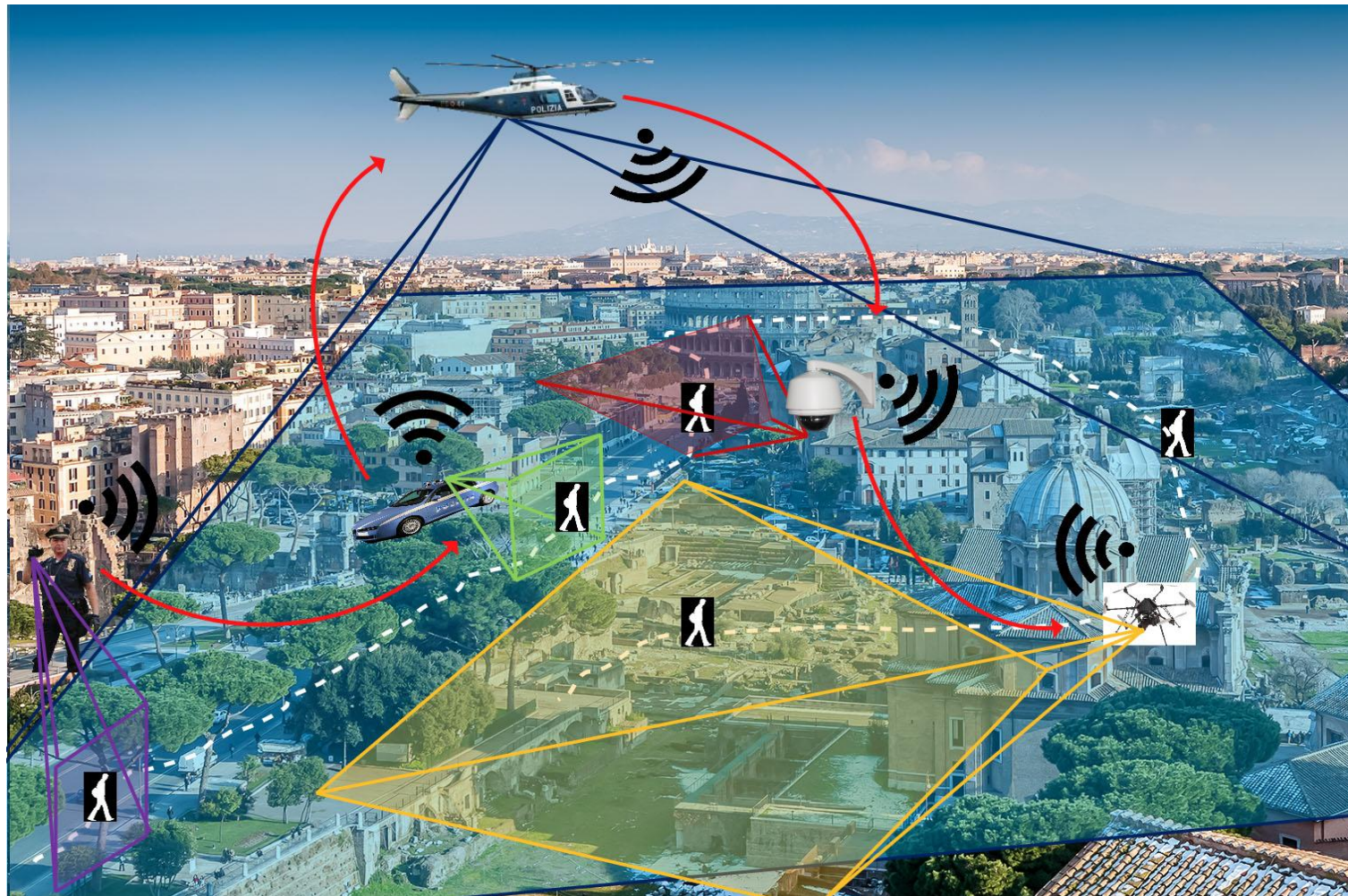
Juan C. SanMiguel

Centre for Intelligent Sensing
Queen Mary University of London

Outline

- Introduction
- Existing simulators
- Basics
- Description (components and extensions)
- Hands-on (installation, GUI and running an app)
- Conclusions

Introduction: why simulators for Camera Networks



DESIRES

Cameras

- Sensing
- Processing
- Communication
- Battery-powered

...

Network

- Collaboration
- Adaptation
- Scalability

...

Reproducibility

External factors

Existing simulators for camera networks

	Code	Type	Resources	Processing	Comms	Extendable	Focus
OVVV [CVPR2007]	C++ (Win)	3D	-	-	-	No	Virtual worlds
SLCNR* [IEEE JETCAS2013]	C++ (Win)	3D	-	Vision routines	-	Yes	Virtual worlds
CAMSIM [SISO2013]	Java	2D	-	-	Protocols	Yes	Coordination
WSVN** [WMCNC2010]	C++ (Linux)	2D	Battery, clock, memory	-	Wireless, MAC	~	Video monitoring
M3WSN** [Simutools13]	C++ (Linux)	2D	Battery, clock, memory	-	Wireless, MAC	Code not released	Multimedia TX
WiSE-Mnet** [SSPD2011]	C++ (Linux)	2D	Battery, clock, memory	Cameras & trackers	Wireless, dummy	Yes	Camera networks

*Paid license required

**Requires the libraries: Omnet++ and Castalia

Outline

- Introduction
- Existing simulators
- Basics
- Description (components and extensions)
- Hands-on (installation, GUI and running an app)
- Conclusions

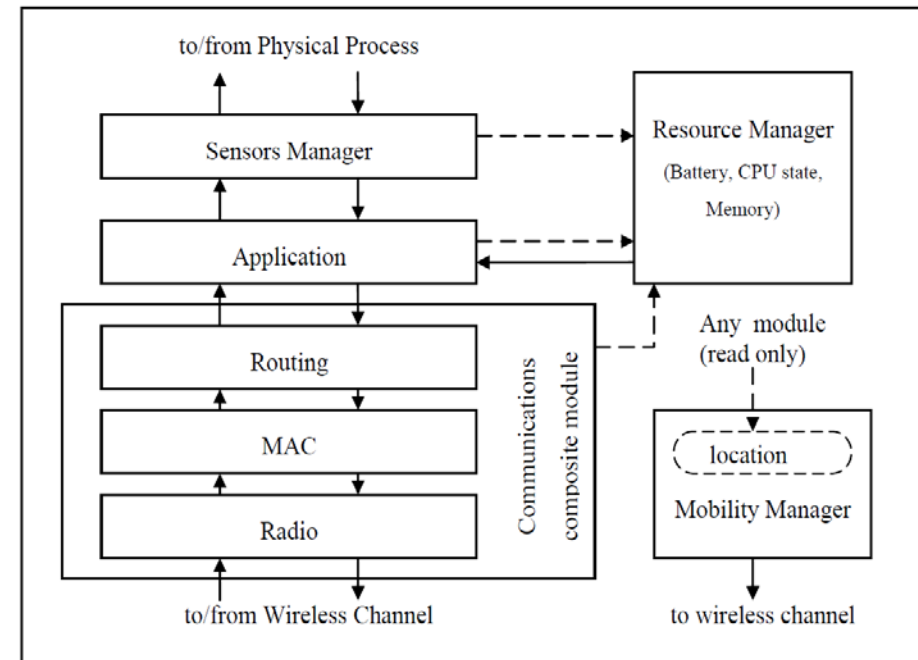
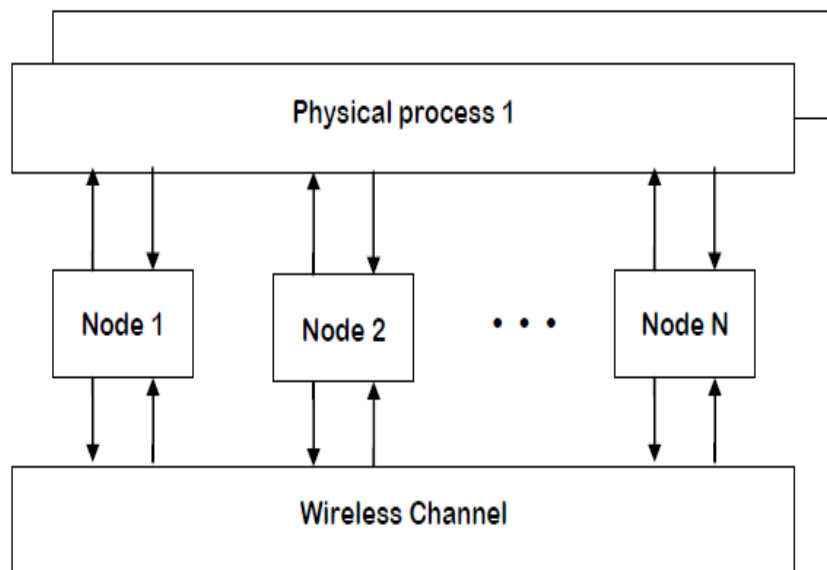
WiSE basics: Omnet++

- Generic discrete-event simulation engine
- Generic modules interactions can be defined
 - behaviour is coded in C++
 - interconnections/composition specified through a Network Description (NED) language
 - parameters can be set through configuration files
- Highly flexible and extensible with external libraries
- Network elements
 - nodes, protocols, channels
 - provided (externally) as simulation models (INET, MiXiM, Castalia)

<http://www.omnetpp.org>

WiSE basics: Castalia

- Wireless sensor networks (WSNs), body area networks (BANs) and networks of low-power embedded devices
- Defines the wireless environment and the node architecture

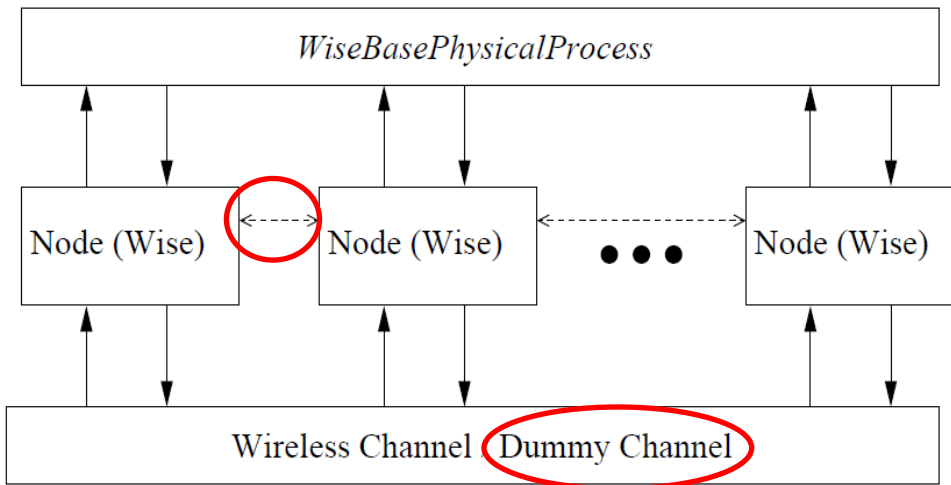


<http://castalia.npc.nicta.com.au/>

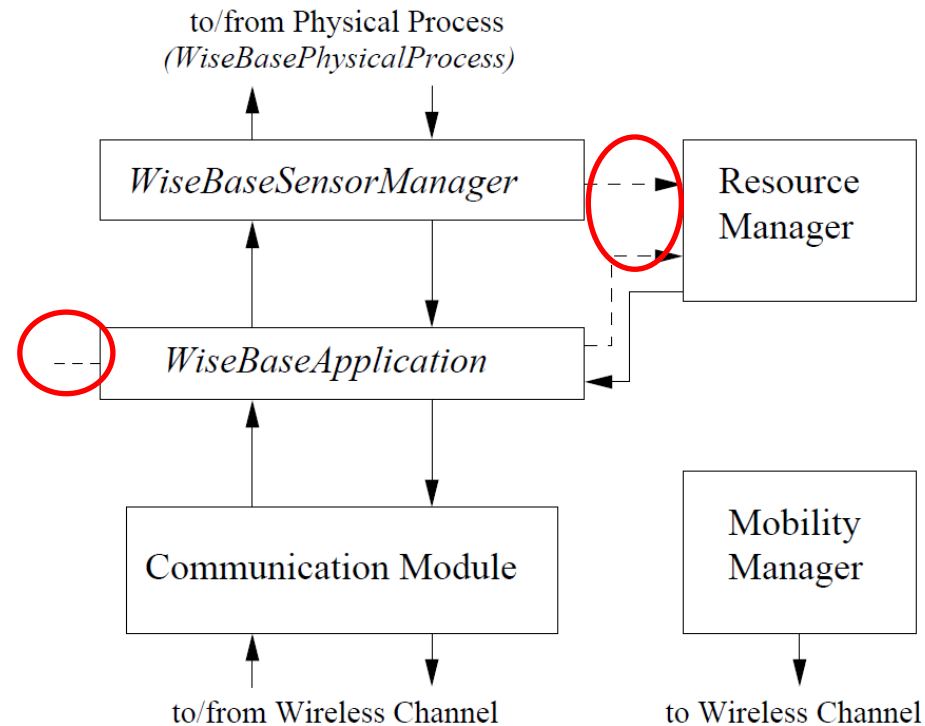
WiSE basics: architecture

- WiSE extends Castalia for Wireless Camera Networks

WiSEXXX files → extensions



(a) Network Model



(b) Node (Wise) Model

WiSE basics: discrete event simulation

- Every sensor/node is independent
- There is no linear script (Matlab) or main (C/C++ projects)
- Omnet++ automatically starts nodes and physical processes
- Communication: message exchange between nodes
- Processing: received messages in discrete units

Tic Toc example

More info at <http://goo.gl/L3SYBo>



Outline

- Introduction
- Existing simulators
- Basics
- Description (components and extensions)
- Hands-on (installation, GUI and running an app)
- Conclusions

WiSE components: NED files

- Describe internal/external connections

```

module Node {
  parameters: //basic parameters
    double xCoord = default (0);
    double yCoord = default (0);
    //...

  gates: //connections from/to other modules
    output toWirelessChannel;
    input fromWirelessChannel;

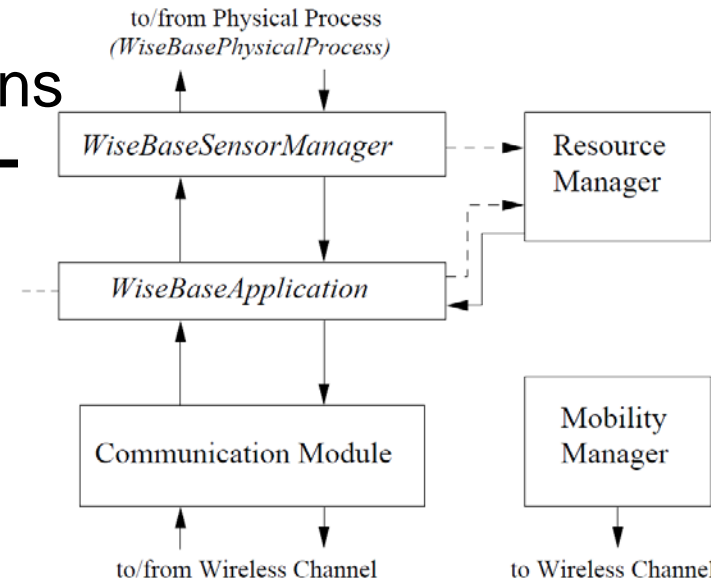
    output toPhysicalProcess[];
    input fromPhysicalProcess[];

  submodules: //submodules of the node
    Communication: node.communication.CommunicationModule;
    MobilityManager: <MobilityManagerName> like node.mobilityManager.iMobilityManager;
    ResourceManager: node.resourceManager.ResourceManager;
    SensorManager: <SensorManagerName> like wise.node.sensorManager.WiseBaseSensorManager;

  connections allowunconnected: //connections between node and submodules
    Communication.toNodeContainerModule --> toWirelessChannel
    fromWirelessChannel --> Communication.fromNodeContainerModule
    Application.toSensorDeviceManager --> SensorManager.fromApplicationModule;
    Communication.toApplicationModule --> Application.fromCommunicationModule
    SensorManager.toApplicationModule --> Application.fromSensorDeviceManager;
    //... |

    ResourceManager.toSensorDevManager --> SensorManager.fromResourceManager;
    //...
}

```



(b) Node (Wise) Model

Source code in
.h, .c and .cc
files

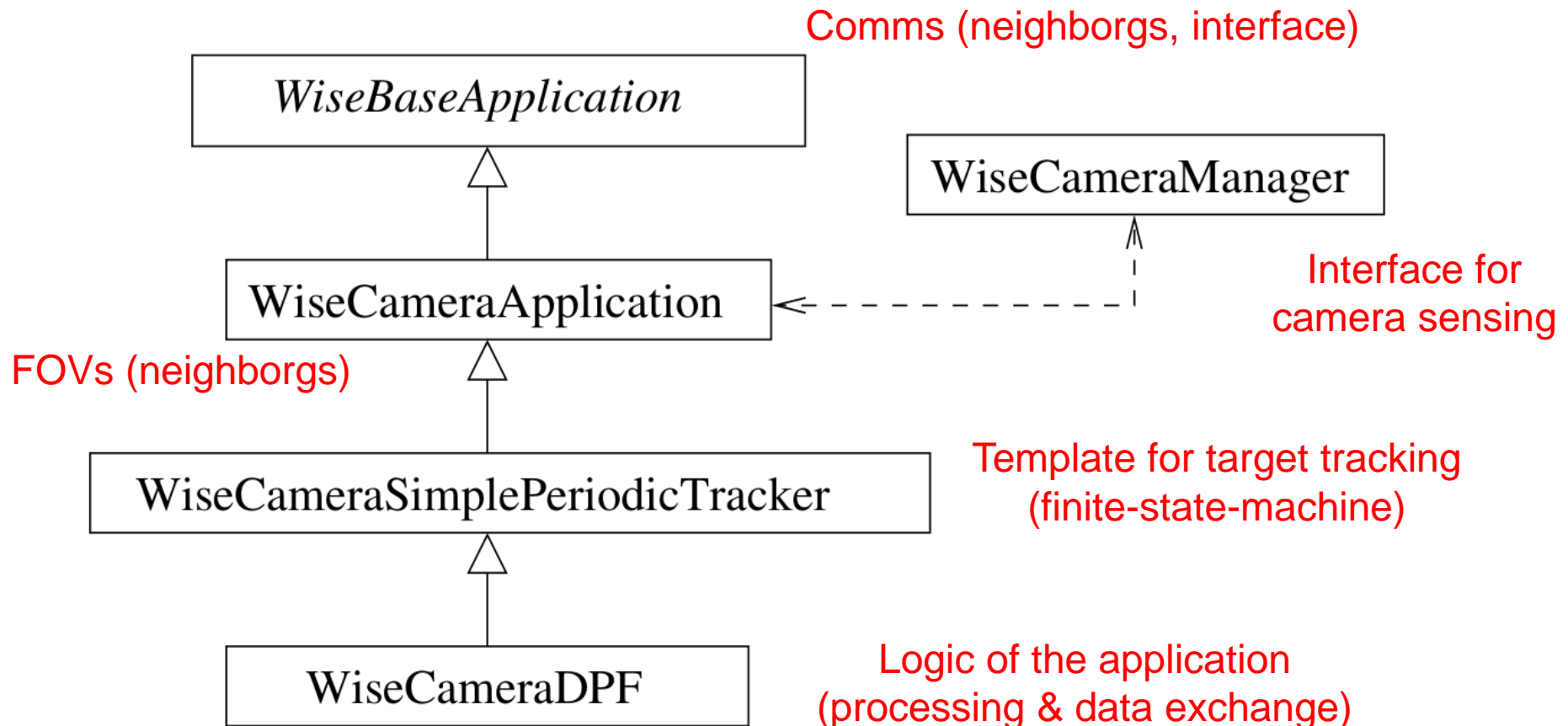
WiSE components: sensing

- Moving targets are represented as "Physical processes"
 - 2D targets --> moving squares (position and size)
 - Type of motion (linear, random,...)
 - Frequency for updating position
- Sensing
 - Return data only if target is within Field of View (but reads all!)
 - *WiseCameraSimplePeriodicTracker* class implements iterative sensing --> frequency to be set

Target position update might be
different to sensing frequency

WiSE components: applications

- Sensor logic (processing, receive/send messages,...)
- Application class hierarchy



WiSE components: application (initialization)

- *Startup()* method
- Set timers to define node's behaviour (e.g., sensing rate)
- Initialize variables

```
#include "WiseAppTest.h"
using namespace std;

Define_Module(WiseAppTest);

#define ALARM_SEND_PACKET 0
#define ALARM_SENSOR_SAMPLE 1
#define LOGGER logger << "[" << simTime() << "]" @ " << self << " : "

ofstream WiseAppTest::logger;

void WiseAppTest::startup()
{
    // This function is called upon simulation start-up
    if (!logger.is_open())
        logger.open("myLog.txt");
    LOGGER << "WiseAppTest::startup() called" << endl;

    // Set alarm to send a packet (0 delay -> NOW).
    setTimer(ALARM_SEND_PACKET, 0);
    // Set alarm to request a sample to the sensor manager (in 8
    setTimer(ALARM_SENSOR_SAMPLE, 8);
}

void WiseAppTest::finishSpecific()
{
    // This function is called when simulation is finishing
    LOGGER << "WiseAppTest::finishSpecific() called" << endl;
}
}
```

WiSE components: application (processing)

**Response to
received message
(on-demand task)**

fromNetworkLayer()

```
void WiseAppTest::fromNetworkLayer(WiseApplicationPacket * rcvPacket,
                                   const char *src, double rssi, double lqi)
{
    // Function called when a packet is received from the network
    // layer of the communication module
    LOGGER << "WiseAppTest::fromNetworkLayer() called" << endl;

    // Print some packet info: sender ID, RSSI, LQI, payload(hex)
    LOGGER << "\tRx from" << string(src) << " with rssi=" << rssi <<
        " lqi=" << lqi << endl << "\tPayload[] = " << hex;
    for (unsigned c = 0; c < 100; c++)
        logger << (unsigned int) rcvPacket->getPayload(c) << " ";
    logger << dec << endl;
    // Calculate application-to-application communication delay and print it
    ApplicationInteractionControl_type ctl = rcvPacket->getApplicationInteractionControl();
    double l = 1000 * SIMTIME_DBL(simTime() - ctl.timestamp);
    LOGGER << "\t app2app delay = " << l << endl;
}
```

**Timer callback
(repetitive task)**

timerFiredCallback()

```
void WiseAppTest::timerFiredCallback(int index)
{
    // Called when an alarm expires:
    LOGGER << "WiseAppTest::timerFiredCallback() called";
    switch (index) {
        case ALARM_SENSOR_SAMPLE:// alarm was for sensor reading:
            // query the sensor manager a new sample (image)
            requestSensorReading();//call the sensor reading function
            break;
        case ALARM_SEND_PACKET:// alarm was a send packet: create a simple packet of 19200
            // bytes, put some payload and broadcast it.
            WiseApplicationPacket * pkt = new WiseApplicationPacket("Test Pkt",APPLICATION_PACKET);
            // set packet details
            // ...
            toNetworkLayer(pkt, BROADCAST_NETWORK_ADDRESS); //send a message to network
            break;
        default:
            // unexpected alarm ID: generate and error
            opp_error("WiseAppTest::timerFiredCallback(): BAD index");
    }
}
```

WiSE components: application (communication)

- Via packets
 - Defined in *.msg files
 - Contains the variables
 - Depends on application
- Send packets to network:
 - Specific nodes
(in WiseBaseApplication.cc)

toNetworkLayer()

- Comms/vision graph
(In WiseCameraSimplePeriodicTracker.cc)

- Channel: wireless (real) and dummy (ideal)

WiseCameraICFMsg.msg

```
cplusplus {{
    #include "WiseApplicationPacket_m.h"
    #include "WiseDefinitionsTracking.h"
    #include <opencv.hpp>
}};

class WiseApplicationPacket;
class nonobject cv::Mat;

packet WiseCameraICFMsg extends WiseApplicationPacket {

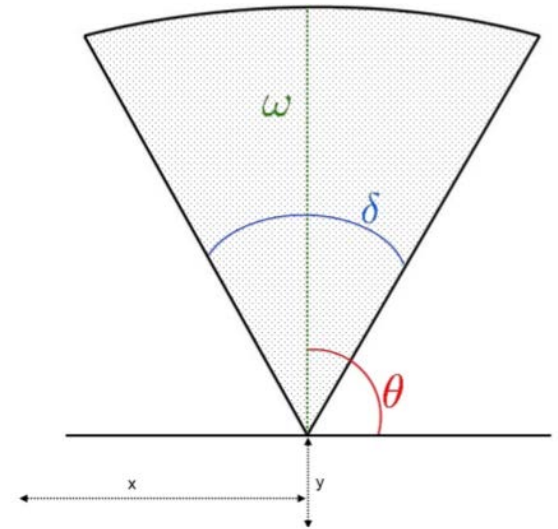
    unsigned long trackingCount;
    unsigned long iterationStep;
    unsigned int targetID;
    unsigned int TypeNeighbour;

    cv::Mat matrix; // OpenCV matrix
}
```

send_messageNeighboursCOM()
send_messageNeighboursFOV()

WiSE extensions

- Capturing from Video files
- Directional sensing (2D FOV)
- Communication/Vision graphs
- Buffer for synchronized comms
- Algorithms
 - Single target tracking
 - Kalman Consensus Filter (KFC)
 - Information Weighted Consensus Filter (IWCF)
 - Multiple target tracking
 - Information Weighted Consensus Filter (IWCF-NN)



Outline

- Introduction
- Existing simulators
- Basics
- Description (components and extensions)
- Hands-on (installation, GUI and creating/running an app)
- Conclusions

WiSE hands-on: installation

1. Install dependencies of Omnet++
2. Install Omnet++
3. Install dependencies of OpenCV
4. Install OpenCV
5. Download WiSE package*
6. Setup a project using the WiSE package*

*Identical installation for Castalia (not required as it is included in WiSE)

Only runs in Linux!!!
(can run in Windows without OpenCV)

WiSE hands-on: GUI

The screenshot displays the OMNeT++ IDE interface. On the left, the Project Explorer shows a tree view of the project structure. A red arrow points to the 'debug' button in the toolbar. On the right, the main editor window shows the source code for 'WiseCameraKCF.cc'. The code includes headers, defines macros for logging, and implements the 'WiseCameraKCF' class with methods for destructor, initialization, and startup.

Project Explorer Structure:

- Castalia-v3.0
 - WISE
 - Binaries
 - Archives
 - Includes
 - bin
 - doc
 - out
 - Simulations
 - src
 - wise
 - Simulations
 - src
 - node
 - wise
 - gui
 - node
 - application
 - sensorManager
 - utils
 - wirelessChannel
 - world
 - SensorNetwork.ned
- 20131128src.zip
- README.txt

- CastaliaBin - [x86_64/le]
- CHANGES.txt
- Doxyfile.cfg
- LICENSE
- LICENSE_2
- LICENSE~
- makeclean
- Makefile
- Makefile.wise
- makemake
- NED_config

Code Snippets:

```
// Copyright (C): Juan C. SanMiguel, 2013

#include "WiseCameraKCF.h"
#include "TMacControlMessage_m.h"
#include "WiseCameraKCFMsg_m.h"

Define_Module(WiseCameraKCF);

#define LOGGER logger << "[" << simTime() << "]" @ " << self << " : "
#define BASE_TRACE trace() << "WISEKCF: "

ofstream WiseCameraKCF::logger;
std::ofstream *final_writer=NULL;

/*! Class destructor */
WiseCameraKCF::~WiseCameraKCF()
{
    if (final_writer) {
        final_writer->close();
        delete final_writer;
        final_writer = NULL;
    }
}

/*! Class initialization and getting of user-defined variables in omnetpp.ini*/
void WiseCameraKCF::at_startup()
{
    // Called upon simulation start-up
    if (!logger.is_open())
        logger.open("myLog.txt");

    if (final_writer == NULL)
        final_writer = new ofstream();

    // Access related-module parameters (read number of targets)
    cModule *network = getParentModule()->getParentModule();
    n_targets = network->par("numPhysicalProcesses");

    // Kalman Filter settings
    procNoiseCov = hasPar("procNoiseCov") ? par("procNoiseCov") : 0.1;
    measNoiseCov = hasPar("measNoiseCov") ? par("measNoiseCov") : 1e-1;

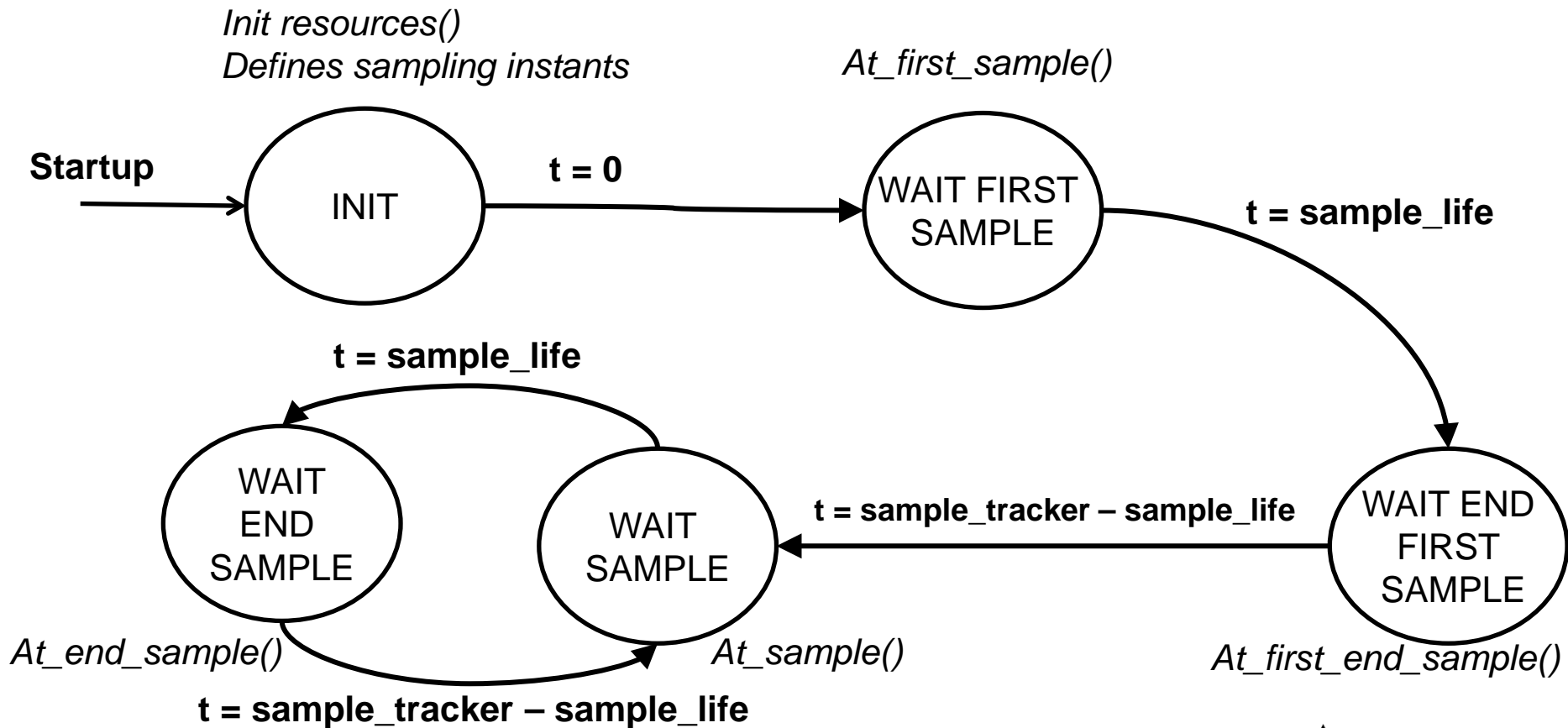
    // Consensus settings
    iter_max = hasPar("iter_max") ? par("iter_max") : 20;
    alpha = hasPar("alpha") ? par("alpha") : 0.1;

    //init structures and lists
    initStructures();
}
```

Code docs
Castalia
Simulations
WiSE code

WiSE hands-on: creating an app (1/2)

- New trackers as derived classes of WiseCameraSimplePeriodicTracker



WiSE hands-on: creating an app (2/2)

```
#include "WiseCameraSimplePeriodicTracker.h"
#include "WiseCameraICFMsg_m.h"
#include "WiseDefinitionsTracking.h" //include for definitions of states and measurements
#include "WiseCameraICF_utils.h" //include specific-structures for single-target tracking of ICF
```

```
#define MAX_SIZE_BUFFER 10
```

```
/*! \class WiseCameraICF
 * \brief This class implements distributed Single-target tracking based on ICF
 */
```

```
class WiseCameraICF : public WiseCameraSimplePeriodicTracker
{
```

```
private:
```

```
    // Define variables
    // ...
```

```
protected:
```

```
    // Functions to be implemented from WiseCameraSimplePeriodicTracker class
    virtual void at_startup(); //!< Init internal variables.
    virtual void at_timer_fired(int index) {} ; //!< Response to alarms generated by specific tracker.
    virtual void at_tracker_init(); //!< Init resources.
    virtual void at_tracker_first_sample(); //!< Operations at 1st example.
    virtual void at_tracker_end_first_sample(); //!< Operations at the end of 1st example.
    virtual void at_tracker_sample(); //!< Operations at the >1st example.
    virtual void at_tracker_end_sample(); //!< Operations at the end of >1st example.
```

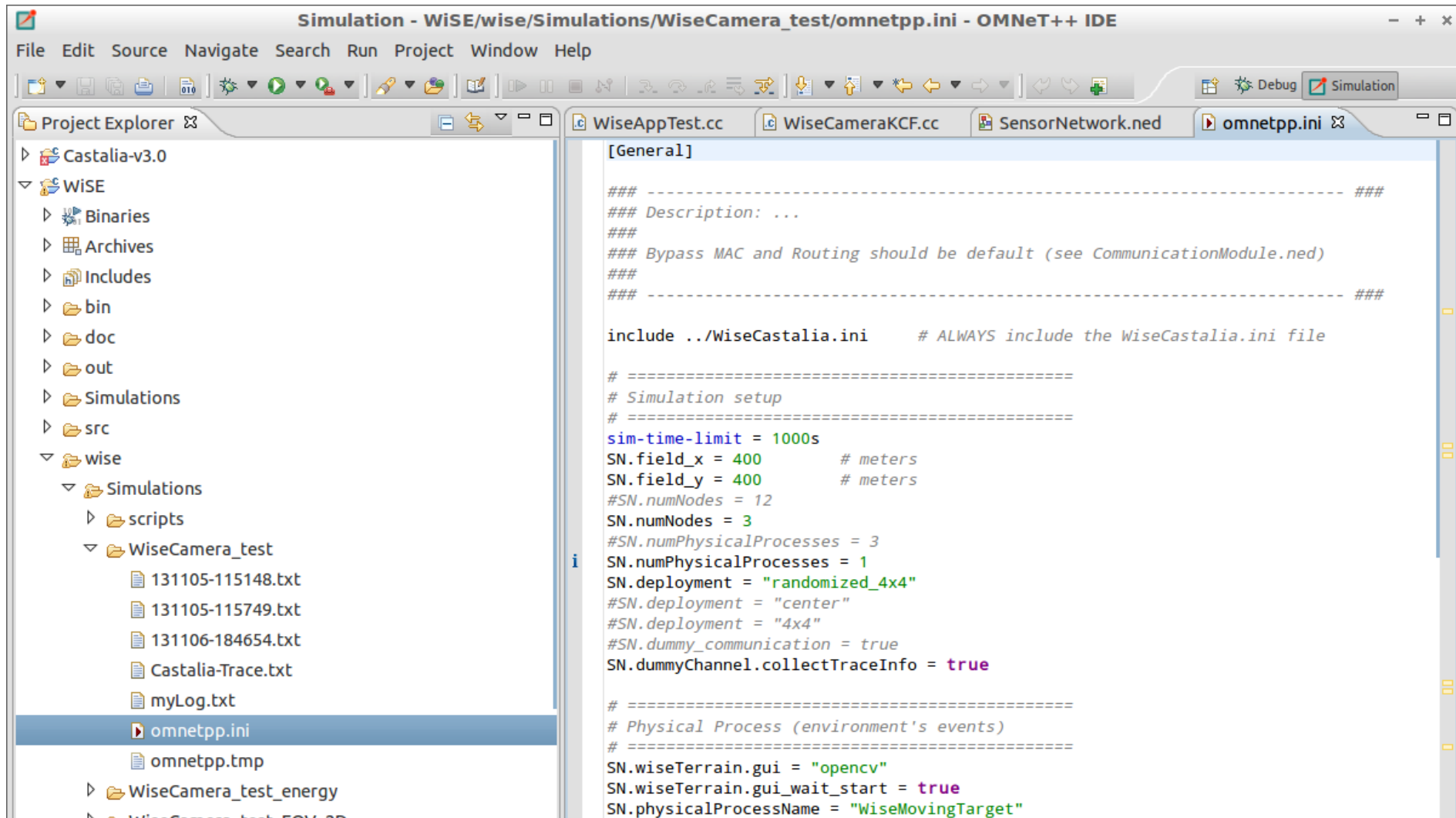
```
    // Functions to be implemented from WiseBaseApplication class
```

```
    virtual void process_network_message(WiseApplicationPacket *); //!< Processing of packets received from network.
    virtual void handleDirectApplicationMessage(WiseApplicationPacket *); //!< Processing of packets received from network
    virtual void make_measurements(const std::vector<WiseTargetDetection>&); //!< Conversion of camera detections into p
    //lists of measurements for tracking.
```

Functions to implement
from tracking template

WiSE hands-on: running an app (1/2)

- Configuration (ini files)



The screenshot shows the OMNeT++ IDE interface. The title bar reads "Simulation - WiSE/wise/Simulations/WiseCamera_test/omnetpp.ini - OMNeT++ IDE". The menu bar includes File, Edit, Source, Navigate, Search, Run, Project, Window, and Help. The toolbar contains various icons for file operations and simulation control. The Project Explorer on the left shows a tree view of the project structure, with the file "omnetpp.ini" selected under the "WiseCamera_test" directory. The main editor window displays the content of "omnetpp.ini", which is an INI file for configuring a simulation. The file content is as follows:

```
[General]

### ----- ###
### Description: ...
###
### Bypass MAC and Routing should be default (see CommunicationModule.ned)
###
### ----- ###

include ../WiseCastalia.ini    # ALWAYS include the WiseCastalia.ini file

# =====
# Simulation setup
# =====
sim-time-limit = 1000s
SN.field_x = 400           # meters
SN.field_y = 400           # meters
#SN.numNodes = 12
SN.numNodes = 3
#SN.numPhysicalProcesses = 3
SN.numPhysicalProcesses = 1
SN.deployment = "randomized_4x4"
#SN.deployment = "center"
#SN.deployment = "4x4"
#SN.dummy_communication = true
SN.dummyChannel.collectTraceInfo = true

# =====
# Physical Process (environment's events)
# =====
SN.wiseTerrain.gui = "opencv"
SN.wiseTerrain.gui_wait_start = true
SN.physicalProcessName = "WiseMovingTarget"
```

WiSE hands-on: running an app (2/2)

The screenshot shows the 'Run Configurations' dialog box in WiSE. The left pane displays a tree view of configurations, with 'WiseCamera_test_FOV_2D' selected under the 'OMNeT++ Simulation' category. The right pane shows the configuration details for this simulation:

- Name:** WiseCamera_test_FOV_2D
- Working directory:** /WiSE/wise/Simulations/WiseCamera_test_FOV_2D
- Simulation:**
 - Executable: opp_run Other: /WiSE/CastaliaBin
 - Ini file(s): omnetpp.ini
 - Config name: (empty)
 - Run number: (empty)
 - Processes to run in parallel: 1
- Options:**
 - User interface: Default Command line Tcl/Tk Other: (empty)
 - Record eventlog: Default Yes No
 - Debug on errors: Default Yes No

Buttons at the bottom include 'Apply', 'Revert', 'Close', and 'Run'. A 'More >>' link is also present.

WiSE hands-on: example

The screenshot displays the OMNeT++ simulation environment. The top window shows a network diagram for a sensor network (SN) with two nodes, node[0] and node[1], connected to a dummyChannel and a physicalProcess[0]. The diagram also includes wirelessChannel and wiseTerrain components.

The middle window shows the source code for the simulation, including configuration files like WiseCameraKCF.cc, SensorNetwork.ned, and WiseCameraKCF.h. The code includes comments and configuration parameters.

The bottom window shows the OMNeT++/Tkenv - SN interface, displaying the current event (Event #31) and the next event (Next: SN.node[1].ResourceManager). The interface also shows a timeline of events, including periodic energy calculation messages, timer messages, and neighbor discover timeouts.

The bottom-most window shows a 3D visualization of the simulation world, with a camera view showing a terrain and a sensor network.

Conclusions

- WiSE enables research on camera networks via simulations of realistic environments
 - Resource constraints
 - Coordination among cameras
 - Real communication protocols
 - Image/Video processing tools
- Expertise required
 - C/C++ language
 - Linux programming skills (gcc compiler)
 - Non-linear design (i.e. collaborative processing)
- Ongoing work: develop resource-limited scenarios

References

OVVV: G. Taylor, A. Chosak, and P. Brewer, “OVVV: *Using virtual worlds to design and evaluate surveillance systems*,” pp. 1-8, CVPR 2007. <http://development.objectvideo.com/>

SLCNR: W. Starzyk and F. Qureshi, “*Software laboratory for camera networks research*,” IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 62(2): 284–293, June 2013. <http://vclab.science.uoit.ca/~faisal/projects/vvs/index.html>

CAMSIM: L. Esterle, P. R. Lewis, H. Caine, X. Yao, and B. Rinner, “*CamSim: A Distributed Smart Camera Network Simulator*,” in Proc. of the IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems Workshops, pp. 1-2, Sept. 2013 <https://github.com/EPiCS/CamSim>

WSVN: A. Pham, C.; Makhoul, “*Performance study of multiple cover-set strategies for mission-critical video surveillance with wireless video sensors*,” in IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communications., pp. 208-216, Oct. 2010, <http://web.univ-pau.fr/~cpham/WSN-MODEL/wvsn.html>

M3WSN: D. Rosario, Z. Zhao, C. Silva, E. Cerqueira, and T. Braun, “*An OMNeT++ framework to evaluate video transmission in mobile wireless multimedia sensor networks*,” in Proc. of the Int. ICST Conf. on Simulation Tools and Techniques, pp. 277–284, Mar. 2013. <http://cds.unibe.ch/research/M3WSN/>

WiSE: C. Nastasi, A. Cavallaro, “*WiSE-MNet: an experimental environment for Wireless Multimedia Sensor Networks*”, Proc. of Sensor Signal Processing for Defence (SSPD), London, UK, 28-29 September, 2011 <http://www.eecs.qmul.ac.uk/~andrea/wise-mnet.html>

References

Implemented algorithms

KFC: Reza Olfati-Saber, J. Alex Fax, and Richard M. Murray. Consensus and cooperation in networked multi-agent systems. In Proceedings of the IEEE, 2007

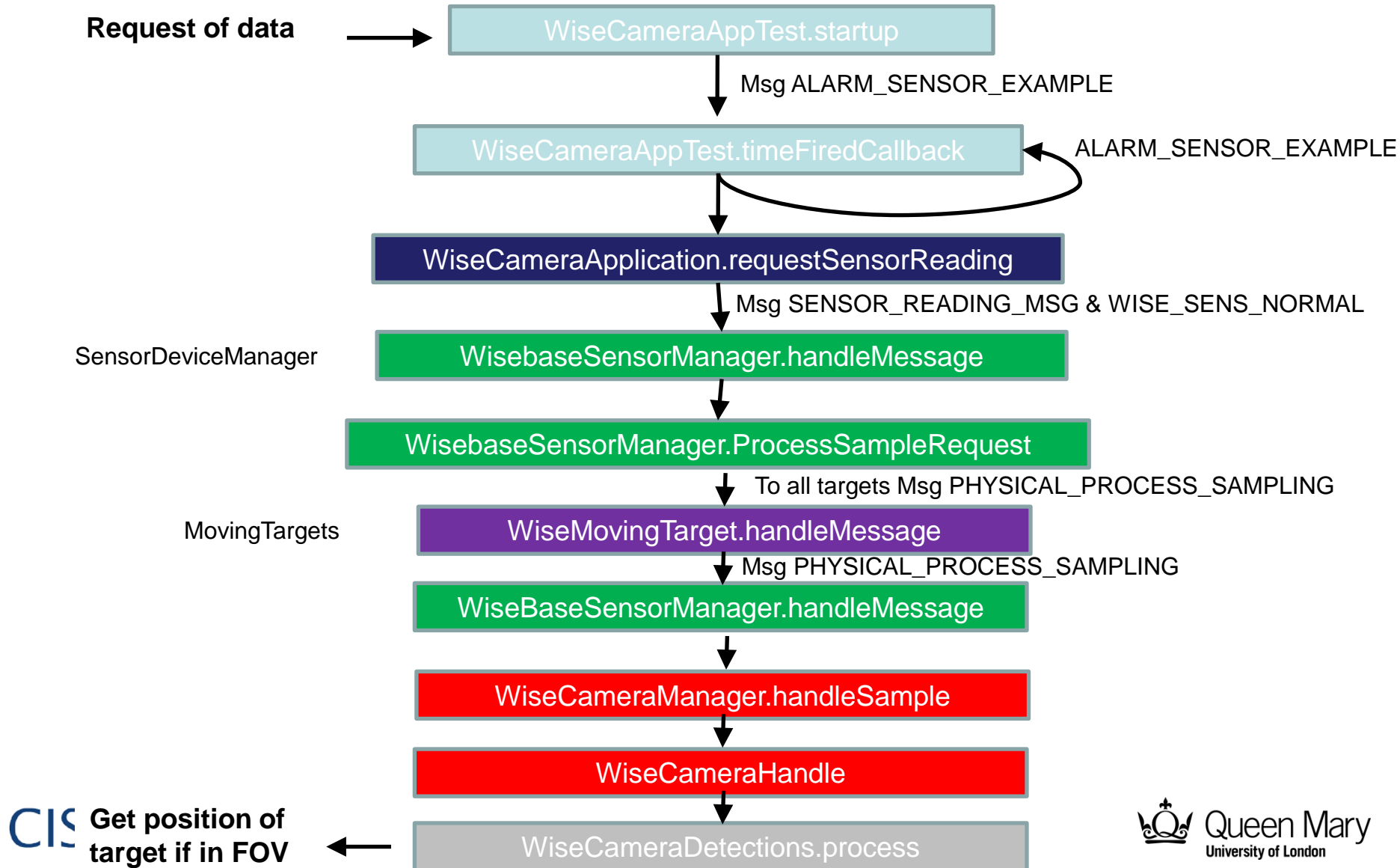
IWCF: A. T. Kamal, J. A. Farrell, A. K. Roy-Chowdhury *Information Weighted Consensus Filters and their Application in Distributed Camera Networks*, , IEEE Transactions on Automatic Control, 2013

ICF-NN: A. T. Kamal, J. A. Farrell, A. K. Roy-Chowdhury, *Information Consensus for Distributed Multi-Target Tracking*,, IEEE Conf. on Computer Vision and Pattern Recognition, 2013

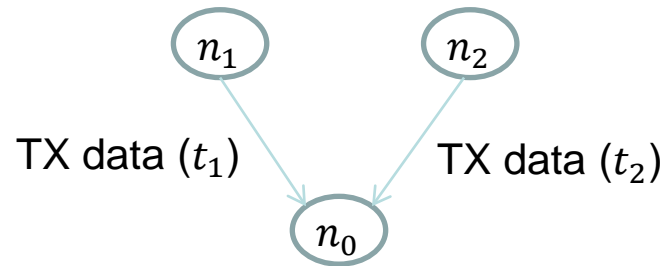
Additional resources: links

- Tutorials Omnet++
 - <http://www.omnetpp.org/doc/omnetpp/tictoc-tutorial/>
 - <http://titania.ctie.monash.edu.au/netperf/netperf-omnetpp-ide-getting-started.pdf>
 - <http://web.univ-pau.fr/~cpham/ENSEIGNEMENT/PAU-UPPA/PROTOCOLES/omnetp.pdf>
- Tutorials Castalia
 - <http://castalia.npc.nicta.com.au/documentation.php>
- Tutorials WiSE
 - <http://www.eecs.qmul.ac.uk/~andrea/wise-mnet.html>

Additional resources: capturing data pipeline



Additional resources: synchronization problem



Implicit coordination strategies require to fuse all neighbour data of the same iteration (ie, iterations of the consensus approach)

→ Data from same iterations received at different time instants ($t_1 \neq t_2$)



Solution (for consensus)

Implement a buffer that stores the data of different iterations. Each node will:

- Save up to MAX_TAM_SIZE iterations of consensus in the buffer
- When receiving data, it will be stored in the corresponding buffer position
- When the last data for an iteration of consensus is received, perform iteration and free the buffer position for other future iterations