

SATIRE for Dummies, ver. 2.0

Raghu K. Ganti
E-mail: rganti2@cs.uiuc.edu

September 25, 2007

Contents

1	Introduction	7
1.1	Hardware Required	8
2	Calibration of Sensor Boards	9
3	Programming the System	11
4	Placement of motes	13
5	Data Collection	17
6	Training and Identification of Activities using HMMs	19
6.1	Training	19
6.2	Identification of Activities	20
7	Data Representation and Access to Data	21
8	Conclusions	23

List of Figures

4.1	MTS310 sensor board with orientation of accelerometer axes .	13
4.2	Placement of motes in a typical SATIRE jacket	15

Chapter 1

Introduction

SATIRE, Smart AtTIRE, is a wearable personal monitoring service that allows users to record their daily activities using a set of motion sensors (accelerometers). Our current implementation uses MicaZ [1] motes and MTS310 sensor boards. MicaZ motes are off-the-shelf hardware devices that have communication, computation, and storage capabilities. Sensing capabilities can be added by attaching appropriate hardware. These MicaZ motes are embedded in a heavy winter jacket¹. During the period when the user wears the jacket, it collects data pertaining to the user's activities, which are recorded in the flash memory of the MicaZ motes. The data thus collected are uploaded wirelessly to an *access mote* (a mote attached to a PC), which is used to identify the activities of the user over the course of the time when the jacket was in use. A complete in-depth overview of the SATIRE system and architecture is described in [3].

This document provides a tutorial on how to install and use the SATIRE service. This includes documentation on how to program the MicaZ motes, how to place and orient these motes in the jacket, how to collect data when the system is operational, and how to identify the activities performed by the user. In what follows, we assume that the user is fairly knowledgeable about programming the MicaZ motes using the TinyOS operating system and development environment. The following section provides a listing of the hardware required for the SATIRE system to be operational.

¹A heavy winter jacket is used to embed the motes unobtrusively due to the form factor of MicaZ motes.

1.1 Hardware Required

The hardware required is as follows:

1. 6 MicaZ motes
2. 5 MTS310 sensor boards, plugged into the MicaZ motes
3. A heavy winter jacket, with sufficient padding
4. 12 AA batteries for operation of MicaZ motes
5. 1 MIB510 programming board
6. 1 serial cable
7. 1 PC/laptop TinyOS compatible

Please refer to Crossbow's website [1] for availability of these devices.

The software accompanying this document is broadly divided into two directory trees, the first is the TinyOS operating system tree, in the directory called `tinuos-satire`. The second pertains to human activity identification, and is in the directory called `satire-hmm`. Each directory has a `README` file of its own that explains how to install and configure the software. We strongly suggest that you read these `README` files before proceeding any further.

This document is further divided into six chapters. Chapter 2 describes the method adopted to calibrate the accelerometers on the MTS310 sensor boards. Chapter 3 describes the method adopted to program the motes for data collection. Chapter 4 pictorially describes the placement of the motes in the jacket. Data collection procedures are explained in Chapter 5. The techniques followed to train and identify human activities using Hidden Markov Model (HMM) based algorithms are explained in Chapter 6. Chapter 7 describes the standard data format representation of the sensory data collected and the activity information identified. Pointers are provided to tutorials about accessing `mysql` database using C and Java. Finally, we conclude this document in Chapter 8.

Chapter 2

Calibration of Sensor Boards

This chapter provides an explanation as to why we need to calibrate the accelerometers on the MTS310 sensor boards and how we go about doing so.

The accelerometers on different sensor boards provide different readings for the same orientation. For example, when two motes (with sensor boards attached) are placed on a horizontal surface with the X and Y axes oriented in the same direction, they provide readings that are widely different. This suggests that the accelerometers need to be calibrated with respect to a standard value (zero g in this case).

To do so, the experimental setup is as follows. Two motes are required, with one mote attached to the PC through the MIB510 programming board (and the serial cable), and the other mote with the MTS310 sensor board attached. The mote that is attached to the PC is programmed with the TOS-Base code, that is found in `TOSROOT/apps/SatireJacket/TOSBase`. Tutorial on TinyOS, installing TinyOS programs on motes can be found at [2]. The mote that has the MTS310 sensor board attached is programmed with the Calibrate code that can be found in `TOSROOT/apps/SatireJacket/Calibrate`.

After installing the code, the user needs to start the ListenRaw program that can be found in `TOSROOT/tools/java/net/tinyos/tools/` directory. This code is executed from the `TOSROOT/tools/java/` directory with the following command:

```
java net/tinyos/tools/ListenRaw -mica2 COM1, where COM1 is the port to which the MIB510 programming board is attached to.
```

This program listens to the serial port and outputs the data to STDOUT. The data that is output to STDOUT needs to be redirected to a file, which can be done by the output redirection provided in any shell or using the

”tee” command. For example, if you want to collect the data into a file called `output.dat`, then it can be done via one of the following commands:

```
java net/tinyos/tools/ListenRaw -mica2 COM1 > output.dat
java net/tinyos/tools/ListenRaw -mica2 COM1 | tee output.dat
```

Once the listen program is started, the mote attached to the PC is turned on (if the power to the programming board is connected, then the mote is already turned on and it is not necessary to turn this mote on). Finally, the mote with the MTS310 board is turned on. Before turning this mote on, ensure that the mote is placed on horizontal surface, such that the sensor board is parallel to the flat surface. Data collection will begin at this point, when the mote that has the MTS310 board (the one we are trying to calibrate) will toggle its green LED. When the yellow LED is turned on, data collection will cease, at which point the ListenRaw program needs to be terminated.

At this point, an output file with the data collected is created. To obtain the calibrated X and Y values, the java program `Calibrate` needs to be executed, which takes as input, the output file that was created, as described above. Each of the 5 motes of the SATIRE system needs to be calibrated. The calibrated values need to be entered in the file `satire-hmm/conf/motes.conf`. The old values in this file are to be deleted and the new values obtained need to be entered. This file has three columns, the first column indicates the mote id, the second is the calibrated X value, and the third column is the calibrated Y value. A total of five rows exist in each `motes.conf` file, one row for each mote. It is also essential to ensure that the mote id’s remain the same when the mote with the MTS310 board is programmed later with the SATIRE jacket code. We also **require** that one of the mote is programmed with the id 1.

Chapter 3

Programming the System

This chapter describes the method followed to program the SATIRE system, that includes the programming of the data collection motes and the base mote. We will assume in the rest of the chapter that the user is familiar with programming MicaZ motes using TinyOS. The README file in the top-level directory of TinyOS describes the directory structure and location of the application to be installed in detail. We will explain the process adopted in programming the motes below.

Navigate to the top-level directory in which TinyOS for SATIRE is installed, let us call this directory *TOSDIR*. The SATIRE code is located in *TOSDIR/apps/SatireJacket/*, navigate to this directory. In order to compile the code, the *compile.sh* script needs to be executed. Before executing this script, ensure that the environment variables as described in the README file associated with the top-level TinyOS directory should be set. Once you set the environment variables, execute the shell script, *compile.sh* in this directory. This will compile all the necessary files and create the executable images that are to be installed on the MicaZ motes.

To program the motes, please do the following. Navigate to the *TOSDIR/apps/SatireJacket/DataMote/* directory to install the images for the data collector motes. Program the data motes with the the image in this directory by typing *make micaz reinstall.<moteid> mib510,/dev/ttyS0*. We assume that the programming board used is an MIB510 and that it is connected to COM1. Please refer to TinyOS's tutorial for instructions on how to program the MicaZ motes.

To program the base station mote, navigate to the *TOSDIR/apps/SatireJacket/Base/* directory and install the image by fol-

NOTE: Program one of the notes with the ID 1. It is mandatory for the system to perform correctly.

Following the above procedure.

Chapter 4

Placement of motes

This chapter describes the placement of motes in the jacket. A typical MTS310 sensor board with the 2-axes of the accelerometers is shown in Figure 4.1. A total of five motes are placed in the jacket, with two motes on

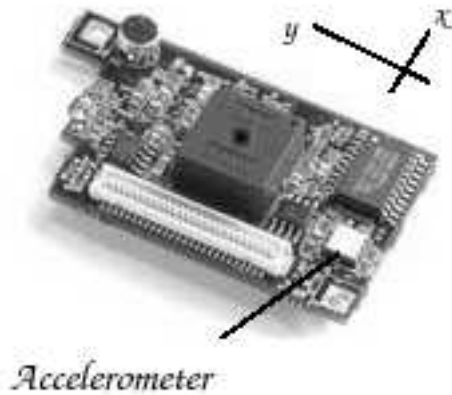


Figure 4.1: MTS310 sensor board with orientation of accelerometer axes

each arm, and one mote in the jacket pocket that is close to the waist. Such a typical placement of the motes including the orientation of the motes is shown in Figure 4.2.

The restriction on placement of the motes in the jacket is minimal. We require that when the person is in a standing position with his arms pointing

downwards, then either the x or the y axis is pointing in the same direction. Also, once the jacket is trained with a particular training data set, it is **necessary** for the motes to remain in more or less the same orientation.

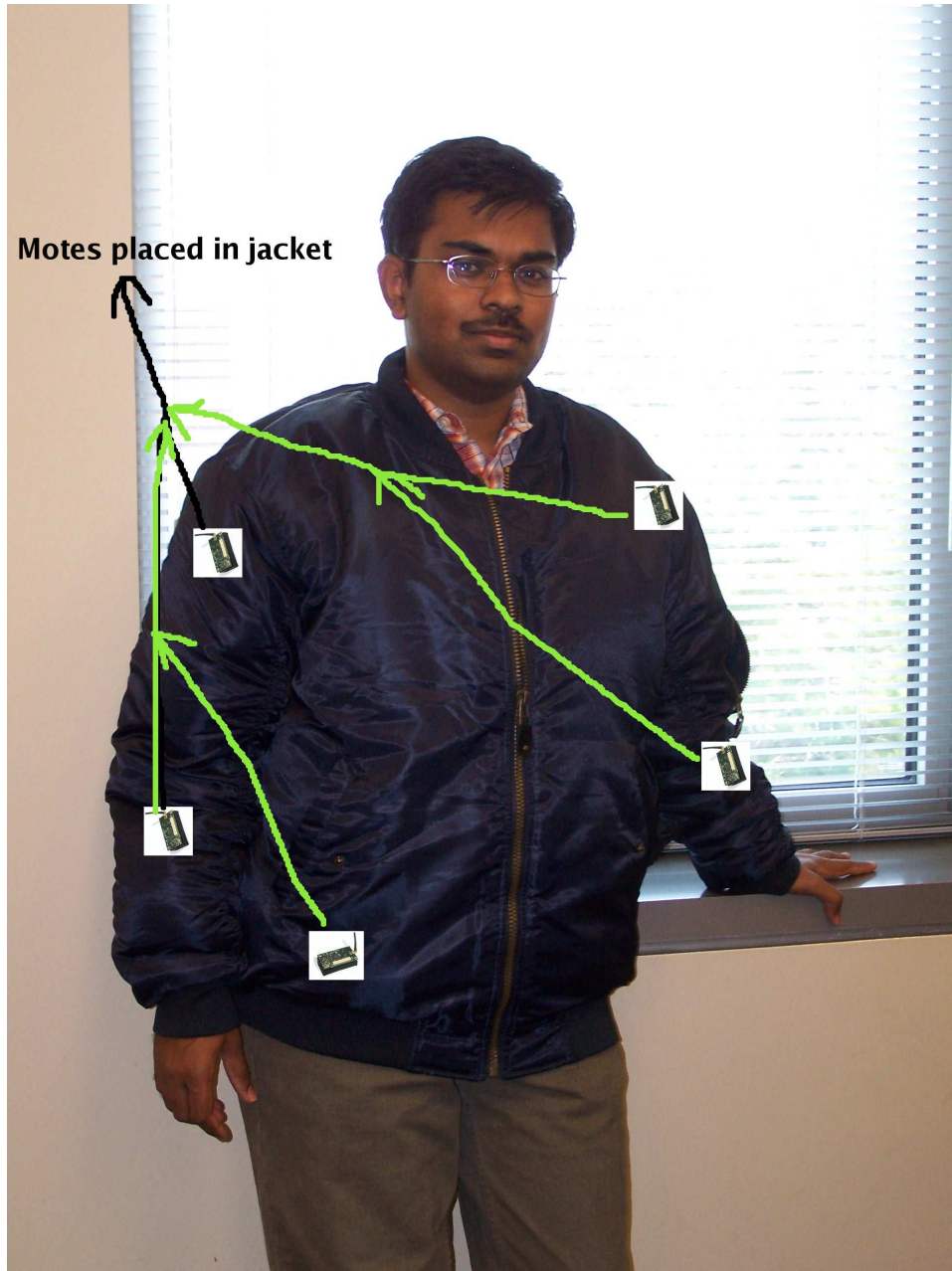


Figure 4.2: Placement of motes in a typical SATIRE jacket

Chapter 5

Data Collection

This chapter describes the experimental setup for data collection from the SATIRE jacket. Once the motes have been properly placed and positioned in the jacket, the motes need to be switched on in a particular order. The mote with id **1** should be turned on last, after all the remaining motes are turned on.

The data collection setup consists of a mote (access mote) attached to the PC/laptop through a MIB510 programming board and serial cable. To begin data collection, the java program `ListenRaw` needs to be started. It is recommended to start the `ListenRaw` program before turning on the access mote to avoid any loss of data. Similar to the setup explained in Chapter 2, the data collected needs to be redirected to an output file. For example, to collect data from a single experiment, we do the following:

```
java net/tinyos/tools/ListenRaw -mica2 COM1 | tee output.dat
```

where `output.dat` is the filename into which the data is to be collected, and `COM1` is the port to which the access mote is connected to the PC. To end the data collection for a particular experiment, terminating the `ListenRaw` application will do so. The raw hexadecimal data collected is now in the file `output.dat`.

Once the raw data has been collected, it needs to be processed in order to generate meaningful data. To do so, please follow the instructions in the README file associated with the `satire-hmm` high level directory.

Chapter 6

Training and Identification of Activities using HMMs

This chapter explains how to go about training the jacket to a specific user and using this trained data to identify future activities of that user. Training is necessary for identifying a particular user's activities due to the machine learning nature of our approach, and the fact that different people perform the same activities differently. We divide this chapter into two sections, namely Training and Identification of Activities. The former section discusses how to do training and the latter provides information on how to identify activities.

6.1 Training

The training stage involves developing models for each activity. This can be broadly classified into two stages viz. the data collection stage for each activity and the generation of models for these activities. The user should perform a short experiment (about 5 minutes) during which they perform that particular activity. For example, if data for the activity **sitting** needs to be generated, the user will switch on the jacket ¹ and perform the given activity (in this case, **sitting**). Data collection is done as explained in Chapter 5. After a certain specified time (we **suggest** at least 5 minutes for 15-30 minute experiments, and longer time periods for longer experiments),

¹For now, switching the jacket on is by turning the notes physically on. We will be soon providing a better mechanism to turn the notes on/off

the data collection program needs to be terminated. Once, the data for various activities have been generated, the second stage of this procedure will begin.

During the second stage, the user will execute the java program `GenerateTrainingFiles` from the top level directory of the `satire-hmm` directory. This program takes as input the *name of the person* and the various activity files that have been created (as mentioned above) and generates the script files necessary for the training. Before the actual training begins, the user is expected to copy the files that have the data pertaining to specific activities to the `data` sub-directory. The training can be done by executing the shell script `train.sh` in the `scripts` directory, that is in the top-level `satire-hmm` directory. Training is a one-time process for a given user for a given garment, and we expect that the activity identification will be a much more smoother process after this initial set of procedures.

6.2 Identification of Activities

When data collection has occurred (or is happening), to identify the various activities performed over the past, the user should first copy the output file to which the data has been redirected to the `data` directory. Then, the shell script `identify-activities.sh` can be executed, which takes the output filename as an argument, generating a graphical plot of activity versus time.

The accuracy of identification of activities is an issue, and similar natured activities may be mis-classified. For example, in our current implementation, we observed that for a given user, it was difficult to differentiate between the activities `climbing stairs` and `walking`. We expect a similar result on other users, but it could be that a positive result is yielded on certain users, due to the varied nature of the activities. We are looking into improving the activity identification algorithms, to incorporate more activities, as well as improving the accuracy of identification.

Chapter 7

Data Representation and Access to Data

This chapter describes the standard format of data representation and the ways to access data. The raw sensor data is stored in XML format, with the following XML fields:

```
<who>...</who>  
<when>...</when>  
<source>...</source>  
<sensorID>...</sensorID>  
<sensorData>...</sensorData>
```

In this format, **who** indicates the person wearing the garment, **when** is the time at which the data was recorded, **source** is the source of the sensor data (for eg., mote 1), **sensorID** is the type of sensor (for eg., accelerometer), and **sensorData** is the data recorded. Interested developers can access these files in standard data format and develop new activity identification algorithms.

The output of the identification algorithms can be stored in a MySQL database. The current database schema for activity storage consists of the fields *source*, *who*, *activity type*, *when*, *where*, *how*, and *confidence*. The *source* field indicates the algorithm used for data interpretation. The *who* field indicates the user of the garment, the *activity type* is the nature of the activity performed (such as **sitting**, **walking**, and **typing**) and the start and end times of these activities are given in the *when* field. The *where* field indicates the location (if GPS data is available). The *how* field indicates special characteristics of the activity identified by the interpretation layer (for eg., walking *slowly*). Finally, the optional *confidence* field indicates the

estimate of the probability with which this classification (by the data interpretation algorithm) is correct. We will be providing repositories of activity information, and the information will be available on the smart attire website (<http://smart-attire.cs.uiuc.edu>). Further information of how to access these data is available on the above website.

The `mySQL` database can be accessed via a programming language of your choice. We give pointers to two different ways of accessing the `mySQL` database. For accessing the database using C, a good tutorial can be found [here](#). This tutorial provides an in-depth explanation of how to write `mySQL` programs in C. To connect to the `mySQL` database using Java, one has to use JDBC. JDBC can be obtained from Sun's website ([download JDBC](#)). The following link illustrates a simple example explaining how to connect to a `mySQL` server.

Chapter 8

Conclusions

In this document, we gave a brief overview of the SATIRE software service and how to go about installing this service given the basic hardware and the software that you downloaded. We observed that there are several steps involved in the installation, which include, the calibration of the sensor boards (accelerometers), the placement of the motes in the jacket, data acquisition, training the jacket and identification of the activities performed. The calibration of accelerometers, placement of the motes, and training the jacket are a one-time process. Although it is tedious, we expect that the user will perform this procedure only once in a particular garment's lifetime (in this case, the jacket). It could also be easily possible for a developer to do the calibration and placement of the motes, thus relieving the end user from programming the motes!

The system is still in a prototypical research stage and we are trying to make the installation and usage as smooth as possible for the end user and developer. Our future work includes the development of a graphical user interface for calibration, data acquisition, training of the jacket, and identification of human activities.

Bibliography

- [1] Crossbow technologies, <http://www.xbow.com/>.
- [2] Tinyos programming manual, <http://csl.stanford.edu/~pal/pubs/tinyos-programming-1.0.pdf>.
- [3] R. K. Ganti, P. Jayachandran, T. F. Abdelzaher, and J. A. Stankovic. Satire: a software architecture for smart attire. In *MobiSys 2006: Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 110–123, 2006.