The ATLAS Wildlife Localization System: System Design and Implementation

A thesis submitted in partial fulfillment of the requirements for the Degree of Master of Science by
Oren Kishon

The research work has been conducted under the supervision of Prof. Sivan Toledo

January 2015
Abstract

This thesis presents the design and implementation of the ATLAS localization system, a system for tracking wildlife using Reverse GPS principals. The main advantages of a Reverse GPS system relative to other wildlife localization systems are energy efficiency, low cost, lightweight transmitter tags and high data throughput. Reverse GPS localization is also very challenging from the engineering and computing aspects, because the system is distributed, with sophisticated receiving basestations at multiple locations, servers, and transmitting tags, and because the system uses of several sophisticated algorithms. Some of the algorithms are sophisticated and complex but not new (e.g., arrival time estimation and nonlinear minimization to estimate locations), but scheduling algorithms and data structures in ATLAS are novel. The system has been deployed in the Hula Valley in northern Israel, currently with 5 receiving stations and has been tested with transmitter tags attached to several different species of wild animals. Localization results have been delivered to the consumers of the system, a group of wildlife biologists.
Acknowledgments

Many people have been collaborating for over two years to develop and deploy ATLAS. This thesis would not have been possible without them. Electrical Engineering Profs. Tony Weiss and Arye Yeredor contributed to the system specification and design, mainly in the areas of signal processing and localization algorithms. MSc student Ronny Ziss working with Prof. Weiss contributed a prototype of the detector and correlator signal processing algorithms. MSc student Bruria Berger, also working with Prof. Weiss, analyzed and simulated in her MSc thesis the system and predicted its performance. Life Scientist Prof. Ran Nathan from the Hebrew University initiated the ATLAS project and provided requirements and design feedback. Life Scientists Nir Sapir, Yoni Vortman, Yotam Orchan, and Yoav Bartan performed and/or assisted in field trials and deployments of receiving stations. My advisor Prof. Sivan Toledo specified the hardware for ATLAS and designed custom hardware components and participated in the design and implementation of the software.

I would like to thank Prof. Sivan Toledo for his guidance throughout my Master degree, both in our joint academic work, my computer science studies and the general life advices he gave me. I would also like to thank Aviad Zuck for letting me participate in his academic work and to him and Alex Druinsky, my lab partners, for the nice time we spent under the fluorescent lights.
# Table of Contents

1: **Introduction**  
1.1 The ATLAS System .................. 4  
1.2 From Engineering Principles to a Working Tool for Ecological Research .................. 5  
1.3 The Scope of this Thesis ........... 6  

2: **Tags**  
2.1 The Tag Database and Virtual Tags ........ 9  
2.2 FSK Inversions and Sequences .......... 11  
2.3 Selecting Frequencies and Transmitters .......... 11  
2.4 Tag Hardware and Firmware ............ 13  

3: **Communication**  
3.1 Physical Links .................. 15  
3.2 Remote Access to Base Stations ........ 16  
3.3 Security .................. 17  
3.4 Serialization .................. 17  
3.5 Fault Tolerant Persistent Queues ........ 19  
3.6 Cellular Modems .................. 21
4: Base Stations

4.1 Base Station Hardware design ........................................ 22
4.2 Base-Station Software: An Overview ............................. 25
4.3 The atlas-radio-server ................................................. 25
4.4 Automatic Gain Control .............................................. 26
4.5 Scheduling Digital Signal Processing Tasks in ATLAS Base Stations 27
4.6 Digital Signal Processing and Buffer Management ............... 30
4.7 Using the Sign of Correlation Peaks ................................ 33
4.8 Remote Maintenance of Unattended Base-Stations ............... 34

5: Localization process

5.1 TOA Algorithm Overview ........................................... 36
5.2 Estimating the Arrival Time of Known Signals ...................... 39
5.3 Solving the Location Equations ..................................... 40
   5.3.1 Shifting and Scaling ............................................ 41
   5.3.2 Eliminating the Time Unknown ................................ 41
   5.3.3 Linearized Expressions ....................................... 42
5.4 Source of Errors in TOA Localization ............................... 45
5.5 Coordinate System .................................................... 46

6: Main Server

6.1 Processing Incoming Detections .................................... 49
   6.1.1 Receiving Input from Remote Stations ....................... 49
   6.1.2 Grouping Together Different Detections of a Single Transmission ........................................ 50
   6.1.3 State Machine of a Detection Group .......................... 51
   6.1.4 Matching a Beacon Transmission .............................. 52
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1.5</td>
<td>Calculating Beacon-Relative Detection Times</td>
<td>53</td>
</tr>
<tr>
<td>6.1.6</td>
<td>Sending Localization Tasks to the Localization Algorithm and Receiving Back Locations</td>
<td>55</td>
</tr>
<tr>
<td>6.1.7</td>
<td>Sending Locations to Consumers</td>
<td>56</td>
</tr>
<tr>
<td>6.2</td>
<td>Database</td>
<td>56</td>
</tr>
<tr>
<td>6.2.1</td>
<td>The Detections Table</td>
<td>57</td>
</tr>
<tr>
<td>6.2.2</td>
<td>The Localizations Table</td>
<td>57</td>
</tr>
<tr>
<td>6.3</td>
<td>Simulator Program</td>
<td>57</td>
</tr>
<tr>
<td>7:</td>
<td>System Deployment and Maintenance</td>
<td>60</td>
</tr>
<tr>
<td>8:</td>
<td>System Configuration</td>
<td>62</td>
</tr>
<tr>
<td>9:</td>
<td>Experimental Results</td>
<td>64</td>
</tr>
<tr>
<td>9.1</td>
<td>Receivers Clock Errors Analysis</td>
<td>64</td>
</tr>
<tr>
<td>9.1.1</td>
<td>Clocks Accuracy</td>
<td>64</td>
</tr>
<tr>
<td>9.1.2</td>
<td>Clocks Frequency Accuracy and Stability</td>
<td>65</td>
</tr>
<tr>
<td>9.2</td>
<td>Localization Field Results</td>
<td>69</td>
</tr>
<tr>
<td>9.2.1</td>
<td>Standard Deviation of Localization of a Stationary Tag</td>
<td>69</td>
</tr>
<tr>
<td>9.2.2</td>
<td>Visualization of Wildlife Tracking</td>
<td>70</td>
</tr>
<tr>
<td>9.3</td>
<td>Simulation</td>
<td>70</td>
</tr>
</tbody>
</table>

References                                           74
1 Introduction

Wildlife tracking systems that are used to gather data about movement of animals in the wild are central to the field of Ecology. Location is one type of data that can be collected by automatic systems. Other kinds of measurements, referred collectively as Biotelemetry could be, for example, heart rate and blood pressure [5]. Advances in technology improve existing tracking technologies but great challenges still remain. The main challenges in tracking systems are cost, localization accuracy, data rates and the weight of tags[3, 4].

The first tracking systems, which are still widely used today, are not automatic systems but rather handled receivers carried by researchers in the field [19]. The manual receivers detect radio-frequency emitting tags attached to animals and typically convert the detected signal into an audible sound for the receiver holder to hear. A single person can find the direction to the animal using a directional antenna and locate it if necessary; a group of researchers determining the direction simultaneously from different locations can estimate the location by triangulating the directions. This labor-intensive method can clearly be replaced by an automatic system. All of the automatic systems that track individual animals are based on tags attached to the animals. The conceptual differences between tracking systems are in the type of signals used, the emission source of the signals and the method of inferring the location out of the detected signals. The more technical differences between systems are in the data retrieval method, the life time of a tag, the frequency of localizations, the spatial accuracy and error estimation and the cost of system components: tags, receivers, computation, maintenance. Detailed comparison tables of existing tracking systems can be found in [3, 14].

GPS systems are based on receiver tags which calculate their own location using times of arrival (TOA) of signals coming from satellites. Their main advan-
tages are their accuracy, in the order of meters and less, their available satellite infrastructure and their global coverage [4, 3, 14]. The disadvantages of GPS tracking stem from the fact that the tag itself is the system’s component that calculates the location. It means that beside the receiver, the tag needs to have a suitable processor, which also consumes more energy. It also means that the locations must be retrieved out of the tag. Transmitting the locations to some other receiver costs processing power and energy. Logging the data requires an extra storage component and it also requires that the tag be located and retrieved. These characteristics of GPS systems make good tags expensive (often over a thousand of US Dollars). The complexity of the tags also make them too heavy for most bird species, despite advanced miniaturization technologies. ICARUS is a global initiative for creating a GPS tracking system that will deal with the above problems but it is still under development [20, 1].

Another type of satellite-based system is ARGOS [7], based on Doppler shift measurements. The satellites calculate the location of transmitting tags based on the change in the frequency of the received signals, caused by the rapid movement of the satellite. Unlike GPS, the location calculation is done in the satellites and transmitted to ground stations, and so it is available to data consumers. However, its spatial accuracy is in only hundreds of meters and often much worse [6]. Other types of tracking systems, less relevant to the system we describe in this thesis, are based on solar geolocation, cellular phone networks and weather radars. They are described and compared in [3].

Systems which are more related to ours are systems which included development of a network of automatic receiver stations in a relatively small area. The purpose of such a system is to concentrate in a specific area of interest (unlike the global GPS systems) in which it is easier to construct a tracking system with accurate, lightweight, energy efficient tags. In contrary to GPS, the costly investment is in the one time infrastructure deployment while the tag units are low-cost. The relatively small range of operation allows the tags to transmit low power signals and hence live longer and to operate in a higher rate. Kays et al. [9] have developed an automatic tracking system based on Angle Of Arrival (AOA). The angle is calculated in each station by using the signal strength, a calculation which requires an array of antennas in each station. The system was deployed and tested in
a rain forest in Panama. The tags used were light weight, but the reported spatial accuracy of the system was considerably lower than of GPS (approximately 50 m).

A different approach to regional automatic wildlife tracking is the “Reverse GPS” principle [10], which is also used by our system. The idea of Reverse GPS is that the tag emits signals which are detected by a surrounding network of receivers. A main server collects the detections which specify the times of arrival (TOA) of the signal at every receiver and computes the tag location. The computation is similar to that performed by a GPS receiver. The receiver stations need to report the time of arrival of tag transmissions, so unlike in the angle of arrival or Doppler concepts no other signal processing is needed other than the detection of the signal. It does, however, requires tight clock synchronization between the stations, since the calculation is based on their time reports. The first Reverse-GPS wildlife tracking system of MacCurdy et al. [10, 11] was tested in the US and in Holland and has GPS-like accuracy.

1.1 The ATLAS System

ATLAS, Advanced Tracking and Localization of Animals in real-life Systems, is an automatic tracking system intended to be deployed in an area of about 10x10 kilometers, based on a self-assembled infrastructure of receiver stations. It uses the Reverse-GPS principle and has many similarities to the system of [10]. Compared to [10], this thesis describes in much more detail the software design of the receiver stations and the main server. One of the differences is that their system’s receiver stations have custom-developed single board computers while we use common Linux PCs which are bigger in size and are meant for general purpose computing but easier to setup and deploy. Our system’s receivers’ clocks are GPS disciplined, the same as in [10]. While their analysis of clock synchronization showed acceptable time errors, our analysis highlighted the need for using beacon tags to synchronized clocks. This concept has significant influence on the design of the main server. Another important difference from [10] is the amount of experiments. While they report about a few experiments done to test the system’s capabilities, the ATLAS system has been fully deployed and operating in
the Hula Valley, north Israel, for more than half a year (one receiver station has been deployed there for more than a year and a half). Our experiments, including the ones described in this work, are based on hours of continuous data. During this long operational time the maturity of the system’s operation and maintenance has evolved greatly. Also, the system was already used for movement ecology experiments on different kinds of animals.

1.2 From Engineering Principles to a Working Tool for Ecological Research

A schematic diagram of the ATLAS system is shown in figure 1.1. In order to describe the system in general, we will begin from the point where a signal is transmitted from a tag and finish where the location report of the tag reaches a consumer of the system. A tag transmits a signal, usually once a second (the period is configurable). The transmission signal codes are unique, and the codes of all tags are known to the receiver stations - this is how the signals are used to identify the tags (See 2). Each receiver station correlates the received signal with a set of signal codes. When the correlation is high (See 4.6), it reports to the main server a transmission of a particular tag has been detected and the time of the detection (accurate to within nanoseconds or tens of nanoseconds). The transmitted signals propagate at a constant speed, the speed of light through air. Therefore, each receiver station receives the signal at a slightly different time, depending on the distance between the tag and the receiver. The localization of the tag is based on the differences in the times of arrival of the signal, between the stations.

A receiver station is comprised of a radio receiver connected to a RF antenna mounted on a tower or mast and a PC that controls the receiver. The PC is connected to the Internet, usually using a cellular modem. The PC in each station configures and starts the receiver, buffers the RF data samples and processes them. The stations connect only to the main server, primarily to send detection reports. The main server, located now in Tel-Aviv University, collects the detection reports from all of the stations and processes them to form localization tasks: input for the mathematical localization algorithm, which another server process runs. The main server receives back estimated localizations and supplies them to the con-
sumers of the system. The main server also archives detections and localizations in its database. This summarizes the structure and operation of the system.

Figure 1.1: The ATLAS system. Localization starts with a transmission of a tag (right) and ends with a localization report reaching the consumer (left).

1.3 The Scope of this Thesis

Designing, implementing, deploying, and evaluating a system of this scale requires a team, and indeed ATLAS has been developed by a team. This thesis focuses on my work within the project; it does not aim to document every aspect of it. My work, and hence the thesis, focused on overall system design, on the design of the processes that control the radios, on the design of the localization server and its algorithms and its database. I also have been maintaining the system, starting at a single-station deployment and ending with a production 5-station working system that tracks multiple wild animals (birds, bats, and aquatic mammals). The maintenance efforts led to many improvements in system design.

This thesis focuses on and documents these aspects. Other aspects of the system, such as tag hardware, RF performance, and so on, are beyond the scope of
the thesis and will be (or have been) described elsewhere.
2 Tags

We begin the discussion of the design of ATLAS with its tags. The system also uses beacons. Beacon have a similar design, but they are not as constrained in terms of physical size.

A typical tag in ATLAS transmits a repeating sequence of \( p \) transmissions, one every \( T \) seconds (ATLAS also supports more complex schedules, but these are beyond the scope of this thesis). The parameter \( p \) is the period of the tag and \( T \) is its interval. Each transmission emits a frequency-shift-keying (FSK) signal at a particular frequency, bit rate, and deviation. The bit sequence is characterized by a family or generator, by its index within that family (or by the seed of the generator), and by its length in bits. We currently support three families: 2047 and 8195-bit long Gold codes Gold [8] (padded to a multiple of 8 bits because the transmitter can only transmit a whole number of bytes) and random sequences generated by a Lehmer random-number-generator Park and Miller [13]. There is no preference between the code types. They all have the feature of low cross-correlation between themselves.

At each of the \( p \) transmission slots in a cycle, the tag transmits one of several transmissions; the same transmission can repeat within a cycle. This allows us to transmit one long transmission and several short ones within a cycle. The long one costs more energy to transmit but is likely to produce a better localization. One transmission in the cycle is the one receivers search for; this transmission should be transmitted only once in the cycle, so that receivers can trivially determine the timing and phase of the tag’s cycle, and it should be the one most likely to be detected by receivers. Normally it should be a long high-power transmission (on a relatively quiet channel, if several are used).
Tags are named using a triplet of positive integers: a vendor identification, a product or model identification, and a serial number (for tags with same vendor and product). Figure 2.1 shows the structure of a simple tag.

![Figure 2.1: The structure of a tag and its transmissions.](image)

**Tag 1, 4, 37:**
- interval=1s, period=2 intervals,
- channel-separation=100 kHz, channel 9 at 433.92 MHz

**Transmission A:**
- code-family=RANDOM, code-index=78, length=8192 bits,
- channel=9, power=10mW, bitrate=1Mb/s, 2fsk-deviation=380kHz

**Transmission B:**
- code-family=RANDOM, code-index=79, length=8192 bits,
- channel=9, power=10mW, bitrate=1Mb/s, 2fsk-deviation=190kHz

Figure 2.1: The structure of a tag and its transmissions. The tag name, [1, 4, 37], is composed of the vendor number 1, product number 4, and serial number 37 (within vendor-product pair [1, 4]). Since the interval is 1 second and period is 2 intervals, the tag time-period is T*p=2 seconds. When a base station detects one of the transmissions, <RANDOM, 78> or <RANDOM, 79>, it correlates it to tag [1, 4, 37].

### 2.1 The Tag Database and Virtual Tags

The tags in a system are described in a text file, normally called tags.txt. This file specifies objects of several Java classes: Vendor, Product, Tag, and Transmission. This file pairs between each tag’s name and its transmission codes. When a receiver station detects a transmission code it finds the tag name that this transmission belongs to using the data from the configuration file.

The same file is normally used to configure physical tags and to configure ATLAS’s base stations and server. This ensures that they are all synchronized.

However, we can run base stations with a different tag-database file, and this is useful for assessing the impact of transmission parameters (deviation, bit rate, etc.) on the ability to detect tags. Suppose we wish to evaluate the difference between two FSK deviations, 190kHz and 380kHz (higher deviation should yield better detection). We could configure two different tags, one for each deviation, and test them. This is cumbersome, requiring two physical tags, and the results
will depend not only on the deviation, but also on any other differences between the two tags. For example, one tag could have a slightly higher power output. A better solution is to configure one tag to transmit at both deviations and to provide base stations with a modified tags.txt in which the physical tag is replaced by two virtual tags each transmitting at one deviation and at twice the inter-transmission interval. Base stations treat the two deviations as completely separate tags, allowing us to compare the behavior of the system at different deviations. Figure 2.2 illustrates this technique. Configuring two transmissions of the same tag as two different tags will not affect the system’s work, but it could help the analysis process of the results.

Figure 2.2: Splitting the physical tag shown in Figure 2.1 into two virtual tags.
2.2 **FSK Inversions and Sequences**

ATLAS supports tags that can invert 2FSK transmissions (exchange the frequencies that represent two symbols). Base stations detect the two symmetric transmissions and estimate whether each transmission was inverted or not. We refer to this estimate, which carries one bit of information from the tag to the base station, as the *symbol value* of the transmission. These inversions allow each transmission to carry one bit of information, which ATLAS can use in several ways that we describe in 4.7. We note that the computational cost of detecting the symmetric transmissions and distinguishing between them is almost the same as the cost of detecting only one of the two; base stations discover the transmission’s symbol value “for free”.

2.3 **Selecting Frequencies and Transmitters**

One key decision that we made early on was to use off-the-shelf integrated transmitters or transceivers in ATLAS tags, at least initially. This decision was driven by the availability of integrated transceivers (sometimes even including a microcontroller) for the frequency range that we wanted to use (UHF or perhaps VHF), for the power output that we wanted to use (mostly 10mW, but suitable transceivers capable of outputting 100mW are also available), and with fairly high bandwidths. The selection of frequency and power output ranges was influenced by several factors. One key factor is spectrum allocation. TOA systems require wideband signals. Wide band channels are not available in the VHF range or are expensive to license. In the UHF range, there are several frequency bands around 315MHz, 325Mhz, 434MHz, 868MHz, and 915MHz where license-free transmission is allowed in many countries (typically only some of these bands are available for license-free uses in a given country). The last three bands allow fairly wide band signals; in Israel, wideband signals are not allowed at 315 and 325MHz, so we excluded them. License-free transmissions must use low power; 10mW is allowed in most cases.

The choice of frequency band also affects the likely antenna efficiency and the path loss. The condition for resonance in a monopole antenna is for the element
to be an odd multiple of a quarter-wavelength, \(\lambda/4\). At VHF, say around 150MHz where many wildlife-tracking tags transmit, many animals cannot tolerate a \(\lambda/4\) (50 cm) antenna. This makes it difficult to ensure that the antenna radiates all the RF energy that the transmitter’s power amplifier generates; short antennas are more difficult to match to the transmitter and they tend to be inefficient or to be narrow banded. At higher frequencies, it becomes easier to design and deploy efficient antennas, but path loss increases, causing lower signal-to-noise ratios (path loss increases by 6dB when the frequency doubles). These factors along with regulatory advice led us to choose the 434MHz band, in which license-free of short-range devices (SRD) emitting up to 10mW is allowed in Europe, Israel, and some other regions between 433.05Mhz and 434.79Mhz. This band is used for remote keyless entry to vehicles, garages, and so on in Europe and elsewhere.

Low power is not only a regulatory issue; it also impacts tag weight. High-power transmitters require heavier power sources, usually batteries. This was another reason to accept the 10-100mW limit of integrated transceivers.

In any license-free bands we expected significant interference. ATLAS indeed suffers from interference to some extent. The alternative, to license exclusive spectrum, was too expensive for us but may be feasible elsewhere or in the future. On the other hand, the significance of the license-free bands to electronic-part manufacturers mean that reference designs and supporting parts (filters and baluns, for example) that are available for the license-free bands are not available for other frequency ranges. We took advantage of this factor at least twice in ATLAS.

Integrated UHF transceivers are designed for mostly for remote-control applications (including keyless entry) and for remote sensing. Most of these applications require low data bandwidths. Consequently, many product offerings only support relatively low bandwidths and low bit rates (10s of kHz and 10s of kb/s). Two manufacturers that we are aware of, Texas Instruments and Silicon Labs, offer high-bandwidth UHF transceivers. Transceivers from both vendors support 2FSK bit rates of up to 500kb/s and deviations of up to 760kHz or so. Higher bit rates are available in 4FSK modulation, but this does not increase the effective bandwidth that is available for time arrival estimation. We decided to use Texas Instruments transceivers, mostly because complete transceiver modules for
434MHz were more easily available (and were very inexpensive; many models are widely available on eBay from Chinese vendors).

### 2.4 Tag Hardware and Firmware

The details of ATLAS tag hardware and firmware are beyond the scope of this thesis, but they have been reported in an article [16]. My work on tags has mainly been to evaluate an alternative transceiver, the CC1200.

Figures 2.3 and 2.4, which also appeared in that paper, show a variety of different tags, as well as tags attached to wild birds.

![Figure 2.3: Tag attachment to wild Barn Owl (left, photo Motti Charter) and Common Kestrel (right, photo Ron Efrat).](image)
Figure 2.4: Hand-assembled prototype tags (top row, a 22-by-16mm CC430-based tag on the left and a 19-by-12mm CC1101-based tag on the right) our production tags (middle row, both sides shown, same layout and size as the top-right prototype), with Encunernet [12] tags for comparison (bottom row, both sides shown, 15.5-by-7.5mm).
3 Communication

3.1 Physical Links

The communication links in the system include intercity links in addition to local area connections. The intercity links are links between each base station (Hula Valley) to the server (Tel Aviv) and between the server to the system’s consumers (in Jerusalem). The local links consist of the link between each base station PC to its radio receiver and the local area network links in the server, to the Matlab localization server and to the database.

The biggest challenge concerning communication in the system was the long distance links between the base stations to the server. The base stations’ PCs are positioned in field locations supplied usually with nothing more than electric outlets. The solution was to connect the PCs directly to the Internet using USB cellular (3G) modems. The cellular connections are reliable and remain connected for weeks, although the modem model we used is the standard product given to private consumers by the cellular service supplier for cases of temporary unavailability of wireless local area network.

Among the system’s communication links the one with the highest throughput is the Ethernet link between the radio receiver and the PC in each base station. Its high throughput is due to the fact that it transmits all raw sample data to the PC without any processing. The rest of the system communication links transmit mostly Java objects and database queries.

A detailed description of the communication links (throughput values were measured using Linux utilities Nethogs and iftop):

- Base station PCs (around Hula Valley) to main server (Tel Aviv): WAN
connection using a cellular modem. The base stations send Java objects with detection timings and other status messages. The average throughput is approx. 0.9 KB/sec, measured while there are three tags detected once a second. During a reconnection scenario, when the buffered messages are drained, the throughput temporarily went up to approx. 75 KB/sec. The stations do not receive back any messages, only protocol ACKs.

- **Base station PCs to their radio receiver**: Each base station PC is connected to its radio receiver via an Ethernet cable. The PC receives from the receiver raw data samples in sampling rate of 8 MHz. The transfer protocol is UDP. Since each sample is 4 bytes, the sampling throughput is 32,000 KB/sec. The receiver adds metadata for every packet that increases the throughput. The measured throughput is 32,800 KB/sec.

- **From the main server out**: The server has three types of external connections. One type is the link back to the base stations. No messages are sent back, so the throughput to each base station is about 0.3 KB/sec, which is made of protocol ACKs. A second connection is to the database in the local network and the third connection is to the the consumer of the localizations, the Eco-movement lab in the Hebrew University of Jerusalem.

### 3.2 Remote Access to Base Stations

There is a need to be able to perform manual SSH log-in to the base station PCs via Internet, for reasons of monitoring, version updating, error solving etc. At first, we configured port forwarding at the base stations for WAN outside access. This solution was not good as it depends on the network routers in case the base station is inside an existing local area network. Instead of that, we found a solution in the way of using the existing, constant, port forwarding in the university network. The university has already a few IP addresses which are always available for outside WAN connections. It means that the stations can initiate a remote connection to one of these open IPs. Using the method of reverse-SSH such connections, once opened, can be used to log-in from the university host back to the station. So each base station opens a socket at startup time to the university network using the
SSH -R option. The SSH -R command opens a socket from the base station to the university open host, and also configures a port forwarding rule at the university network so that a designated port in the university network will be forwarded to that remote base station address. The result is that in order to connect from the university host to a remote base station, the connection is to the local host, on the specific base station port. For instance, for a base station that configured port 44101 for itself in the university network, the command to be ran in the university network will be: “ssh -p 44101 localhost”.

The SSH command in the base station, that runs on startup, also adds the argument for keeping the connection alive, because SSH connections disconnect automatically when they are not used.

3.3 Security

All sockets in the system are SSL sockets. There is a RSA key for the server, and a RSA key for a client. All client keys are the same one. The keys are saved in local files in every host running the system (server or client) and so it is not secure if a attacker accesses the file system of one of the system’s components. It is only secure to the level that a “man in the middle” access to a connection could not decode or hinder it.

3.4 Serialization

Serialization is the process of converting an object in memory, which usually has references to data blocks in different memory addresses, to a continuous byte array so it could be stored in a file or sent on a socket. In the same way the opposite process of restoring an object is deserialization.

All of the connections in the system transfer data by using Java I/O streams, accept connections to database, which use the JDBC (Java Database Connectivity) API. The Matlab component also runs the same custom Java socket classes that other clients run (Java classes can be called in Matlab code). Above the socket layer the data is sent using Java’s serialization stream: ObjectInputStream and ObjectOutputStream. The way Java’s serialization works, shortly, is that it saves
a lighter version of the object with only the volatile variables, which are the class members’ values. All of the other static parts of the object (class name, member types, dispatch vector, etc.) are represented as a hash code. Since both sides of the stream hold the same description of the class, there is no need to pass this complete structure between them, only the hash code.

Using Java serialization over remote data streams has its advantages and disadvantages. The main advantage is coding simplicity. The objects sent over the stream are the objects that are used by the program. Without serialization, in order to send data, there is a need to create a packet object in which the data needs to be copied explicitly (The common practice in C language networking programming). Java Serializable interface takes care of implicitly copying all of the data referenced by a serializable object - if it has members who are references, they are also serialized, and so on, recursively. The same goes in the opposite direction of deserialization in the receiver side. Another big advantage of using object streams is that there is no need to design a communication protocol, i.e messages’ headers which include message type, message size, etc, and the appropriate custom massage decoding in the receiver side.

One feature of Java serialization can be looked as an advantage as well as a disadvantage: The objects must be implemented in the same way in both sides, because when an object implementation is modified, its hashing is modified and so the other side will not be able to find it. It means that implementations of classes can not differ even regarding the parts that are not sent over the stream. The advantage of it is prevention of mismatching in packet structure, when decoding a packet does not match its coding procedure. Bugs caused by this kind of mismatching are sometimes hard to detect. The disadvantage is that this forces all system hosts to be updated with the latest changes of the object. In the ATLAS system, for instance, any change to an object used by both server and base stations requires updating each base station, even if the update is in some server related function of the object.

The simplicity of Java serialization can also be regarded as a disadvantage. The developer has no explicit knowledge of the size of the data sent. The fact that no special packet structure is implemented releases the developer from inspecting the composition of the sent objects, and so much unneeded data sometimes is sent.
An issue we have encountered by surprise is that the serialization mechanism has a caching feature which is enabled by default. It is designed for connections which occasionally resend the same objects. When the sent objects are always new, like in our system, this caching mechanism causes a memory leak which was quite hard to find.

Another disadvantage of Java serialization is that the serialized data can only be contained in streams, i.e., FIFO queues. It means that there is no way to maintain sets of objects in their serialized form in memory or in a file in the way that they will be seek-able, only one object is always available to be deserialized. We have encountered this fact as a disadvantage when we wanted to back up objects in a file and have them be deserialized in a LIFO manner. Our solution for this is described in the next section.

3.5 Fault Tolerant Persistent Queues

Queues are essential program components in the ATLAS system, as we use them to implement two key principles: multitasking and data backup. We have implemented a custom queue, the MessageQueue class, which is used by all of the components who need a queue.

**Implementation in general:** The MessageQueue holds two internal queues: A Java LinkedBlockingQueue and a custom file queue. The LinkedBlockingQueue is a capacity-limited memory queue with the features of thread-safety and blocking add/take when the queue is full/empty. The file queue stores each object in a file in the local file system in a LIFO manner. In order to be able to save to a file, the queue can hold only serializable objects, like the messages we use in the system. The relation between the two queues is the following: When an object is added, first there is an attempt to add it to the memory queue. If the memory queue is full, the object is added to the file queue. When there is a request to take an object from the queue, first there is an attempt to take an object from the memory queue. If it is empty than there is an attempt to take an object from the file queue. If that is also empty, than there is a second attempt to take from the memory queue. The second attempt is a blocking one - it will return only when some other thread will add an object to the queue. The queues that exist in the system:
base station detections to be sent to the server, localization tasks to be sent to Matlab localization algorithm from the server, Matlab incoming queue of localization tasks, localization solutions to be sent from Matlab back to the server, localization solutions and other messages to be sent from the server to the consumers.

**Multitasking:** most components in the ATLAS system have the work flow of processing objects that have arrived from some input stream and later sending result objects to some output stream. The input could be a network socket, a memory queue, etc. For example: the base station listener component in the server receives detection objects on a network socket and outputs localization tasks to the localization algorithm input queue. The MessageQueue implements a thread-safe queue and so the queues in the system implicitly implement the locking mechanisms needed in the interfaces between components, in a design pattern of producer-consumer. The MessageQueue is non limited in capacity so adding to it never blocks. Retrieving from the queue is blocking if the queue is empty, until something is added.

**Connection fail-over:** When the ATLAS system works in its normal way, which means there are no current disconnections, no queue should hold more than a few objects. The memory queue is filled up, causing objects to be added to the file queue, only in cases of some error, or a recovery from an error. An example scenario is when a base station disconnects from the server and so its output queue is buffering detection objects in the file system. After reconnection, the queue is drained to the network socket towards the server. In order to save as much data as possible in case of system failure, when an object is written to the file the write is synchronous, meaning that it is written to the file system immediately memory buffering. This synchronicity comes at the cost of slowing down the queue operations. From the system administration point of view these file queues have infinite capacity: since the local hard drives have hundreds of gigabyte of storage space, it will take weeks or even months until such a queue will fill up the whole drive. Since system errors are detected within hours or days, this is sufficient.
3.6 Cellular Modems

The choice of using cellular (3G) modems in the receiver stations was due to fact that most of the stations are in an outdoor location or in a utility room without Internet infrastructure. The modems are USB dongles in the size of a few centimeters, which is an advantage from the station construction point of view (no extra communication and power cables needed). In terms of data throughput, we found that they suit the system needs (See section 3.1 for throughput values). The model that we used is ZTE MF180 and its firmware was designed for Windows users for cases of unavailable Internet connection, and not as a full time working modem. In our Linux PCs which use the Ubuntu distribution, the operating system was able to identify and install the modem’s driver without a problem (shows one advantage of using a general use computer in the receiver stations). We did find a problem that became a crucial issue concerning the remote operation of the stations: We found that when inserted to the PC, or after a reboot, the USB device interfaces the OS as a CD-ROM device rather than as a modem. The reason is that in Windows OS the CD-ROM mode launches a connection program which later switches the CD-ROM mode into modem mode. In Linux this mode switch needs to be applied manually from the command line. In addition, after the switch the modem did not connect automatically. That meant that in a case of disconnection, including a reboot of the system, the modem will not try to reconnect. We solved the problem by using a script that is initiated by the operating system at startup (Using the Unix rc.local configuration file). The script runs a loop that mode-switches the device and connects it every time it identifies a disconnection. We found that although these modems were not intended to work for long time periods, they stay connected for weeks without disconnections. The auto-connection script allows us the possibility of rebooting the computers remotely (for development purposes).
4 Base Stations

ATLAS base stations detect tag transmissions, estimate their arrival times (and the value that they carry), and send these detection reports to the ATLAS server.

4.1 Base Station Hardware design

Our design criteria for the base-station hardware were as follows.

- High Performance: Base station radio receivers need to have low noise figure and enough gain to detect transmissions that below the noise floor, high dynamic range to cope with strong interfering signals, and a highly accurate and highly stable sampling rate, since sampling-rate errors increase arrival-time estimation errors.

- High-enough bandwidth to receive wide-band transmissions, which are necessary in order to accurately estimate arrival times.

- Use of well-supported off-the-shelf hardware as much as possible, in order to make the system easy to deploy and easy to maintain.

- Low cost (subject to the other criteria).

Base stations use Ettus Research N200 USRP radios with a WBX daughter board and with a GPS-disciplined reference oscillator. The radio downconverts a slice of the RF spectrum to complex baseband and samples the resulting baseband signal at 100Ms/sec, 14-bits per sample. These samples are sent to a computer over an Ethernet link, usually after decimation. All the radio’s synthesizers, including its
sampling clock, are locked to the GPS-disciplined oscillator, which provides good frequency stability and frequency.

The response of the WBX’s receiver front-end is nearly flat over the entire VHF and UHF ranges, which makes it very susceptible to interference, it is not particularly quiet (noise figure of about 5dB), and it does not have enough gain to detect very weak signals. We have therefore added additional front-end elements to compensate for these deficiencies.

We place a low-noise amplifier close to the antenna, to bring the system’s noise figure to below 1dB. The amplifier we use, L432LNA from Down East Microwave\(^1\) has noise figure of 0.7dB or lower, 17-20dB of gain, very high dynamic range (output P1dB of 19dBm) and single-pole filters at both its input and output. The filter at the input is designed to prevent strong out-of-band signals from distorting weak signals in the LNA itself, but it is not sharp enough to protect subsequent stages. Initially we tried to address this by adding an off-the-shelf connectorized bandpass filter (from Cross Country Wireless\(^2\)) followed by a second L432LNA, but this bandpass filter was also not sharp enough to reject strong interferers.

We eventually designed and built a custom filter-amplifier-limiter unit that processes the signal coming from the L432LNA and produces the signal that drives the WBX and the USRP\(^1\). This unit provides another 10dB of gain or so, sharp filtering using a SAW filter (with a bandwidth of about 5MHz), and limits signals to a level that is safe for the WBX and the USRP. This custom unit does not satisfy our off-the-shelf design criteria, but it was necessary and it represents a relatively small and low-cost part of the overall hardware.

The sharp bandpass SAW filter has a high group delay (about 140ns). The delay is a product of the sharpness of the filter, not of its construction, so it cannot be eliminated. As we shall see below, the delay itself is of no consequence, but time- and frequency-dependent variations in the delay can degrade the localization performance. A filter with a high delay is likely to have higher-magnitude instability than a short-delay filter, but our testing indicates that the delay stabil-

---

\(^1\) http://www.downeastmicrowave.com

\(^2\) http://www.crosscountrywireless.net/filter.htm
ity of the filter that we use is good enough. Manufacturers of low-cost filters do not specify their group delay and certainly not the stability of the group delay, and filter with a specified and flat group delay are expensive (these are essentially custom designs). We have found that this is an area where testing was the only practical way to go, as opposed to careful specification of the filter in the design stage.

Samples arriving from the USRP are processed by a regular PC computer. Detection reports are sent to the ATLAS server through the Internet, either using a wired Ethernet connection or a 3G cellular connection.

Our base stations use vertical omnidirectional vertically-polarized gain antennas, which are common (and hence easy to procure from multiple sources) and easy to install. Their gain is directed toward the horizon. These base-station antennas are not a good match to tag antennas, which are also linearly polarized (1/4 wave whips) but which may be in any position. A circularly-polarized base-station antenna would be a better match, but we have not found a commercial circularly-polarized antenna with a suitable radiation pattern.

The overall structure of base stations is shown in Figure 4.1.
4.2 **Base-Station Software: An Overview**

ATLAS base-station computers run two programs. One program, the atlas-radio-server, controls the USRP radio, collects and buffers all the samples that the USRP send continuously, and sends them on demand to the digital-signal-processing (DSP) program. The other program, the atlas-receiver, schedules digital-signal-processing tasks, performs them to detect tag transmissions and to estimate their arrival times, and sends detection reports to the central ATLAS server.

The separation of these activities (buffering samples and performing DSP tasks) is part of our design. The fact that these two activities run in different processes is mostly an outcome of our design and implementation methodology. We implemented the first prototypes of the receiver code in Matlab and later switched to a combination of Java and C. The fact that the software that collects and buffers samples run in a separate process made the transition relatively easy.

The modular design allows us to replace one of the two programs by alternatives. In the radio server, by using abstraction, we have ported the interface to the hardware so it can be used also with low-cost low-performance radios. These do not support accurate localization of distant tags, but they are useful in detecting nearby tags. An alternative to the receiver program is the spectrum analyzer program which instead of interfacing the main server allows us to visualize interfering signals.

4.3 **The atlas-radio-server**

The atlas-radio-server controls a USRP radio and collects its samples using a C++ driver library called UHD (USRP Hardware Driver), which is the interface to all radios made by Ettus Research. The radio server exports its services though a TCP or Unix-domain socket.

The receiver program connects to this socket and sends the radio server commands to set the frequency of the radio and its sampling rate. The radio server responds by instructing the radio to move to that frequency and to start sampling at the required rate. From that point on, the radio sends samples continuously. The
radio server collects them in a large circular buffer that can hold about 30 seconds worth of samples.

Samples arrive from the radio with an absolute time stamp. When the radio powers up or if the radio does not have a GPS-disciplined oscillator, the time stamps have an arbitrary radio-local reference time. Once the GPS locks, the time stamps are GPS based. The radio server retains these time stamps.

After the receiver program sets the parameters of the radio (it never changes them), it starts asking the radio program for chunks of RF. Each request is for a given number of samples and for a particular absolute sampling time. If the requested samples are available in the buffer, they are delivered to the receiver program. Otherwise, the radio server returns an error indication. The receiver program can also request the most recent samples without specifying an absolute time.

An alternate implementation of the atlas-radio-server interfaces the atlas-receiver to RTL2832U-based DVB-T receivers with a USB interface. These low-cost receivers can filter, downconvert, and sample a slice of the VHF or UHF spectrum. The samples have low 8-bit resolution and the sampling rate is 3.2Ms/s or less, but this is enough to detect ATLAS tags at good signal-to-noise ratios.

4.4 Automatic Gain Control

The radio server is also responsible for controlling the gain of the radio. The most appropriate gain is determined automatically using an algorithm that inspects the maximum absolute sample value over each 10ms period. These 10ms periods are classified into *high periods*, *low periods*, and *normal periods*. High periods are those in which the maximum sample value is close to full scale (32000 or above; full scale is 32767). Low periods are those in which the maximum sample value is below 1/8 of full scale (18dB below full scale).

Every 10 seconds the server evaluates the gain of the receiver. If more than half the 10ms periods in the past 10 seconds are high, the server reduces the gain of the radio by 0.5dB. If this is not the case, there are low periods, and at least half the periods are 3dB or more below full scale, the gain is increased by 0.5dB. This
automatic gain control (AGC) algorithm results in slow gain changes that tend to bring the noise floor to about 18dB below the ADC saturation point, even if the radio experiences occasional saturation (at most half the time). This algorithm allocates about 10 or 11 bits of the dynamic range to the weak signals that we are interested in (up to the noise floor) and about 3 bits to tolerating with interferers whose signal is between the noise floor and 18dB above it. Interference at this level does not reduce the system’s ability to detect weak signals.

Interference stronger than 18dB above the noise floor saturates the ADC and destroys the system’s ability to detect weak signals and to estimate arrival times. However, occasional interference at this level does not reduce the sensitivity of the system, thereby allowing it to detect weak signals as soon as the interference disappears.

Persistent strong interference (saturation more than half the time) causes the system to reduce the receiver’s gain. If the reduction is significant, the system loses the ability to detect weak signals. Prior to the deployment of the custom front end, the system often experienced persistent destructive interference that caused the AGC algorithm to reduce gain by up to 15dB. Once we deployed the new front end, we have not experience similar problems.

4.5 Scheduling Digital Signal Processing Tasks in ATLAS Base Stations

Base stations schedule digital signal processing (DSP) tasks using a fairly complex algorithm that is implemented in the Receiver class.

A base station need to report detections of a collection of tags that might be active in its area. The base station partitions the set of tags into two subsets, the tags for which the phase of the transmission cycle is known and the set of tags for which the phase is not known (By knowing the phase the next transmission times can be predicted). These subsets are called the track-set and the search-set. The track set is a priority queue ordered by the next unprocessed transmission of each tag. We use the term unprocessed transmission to emphasize that the base station needs to attempt to receive the transmission (to perform the necessary signal processing on the appropriate RF samples), but the transmission may or
may not be detected. The tag whose first unprocessed transmission is earliest is ordered first in the track set and so on. The search set is unordered.

The scheduling of receptions is organized around an infinite loop. Each iteration starts with a scheduling decision that returns a `CorrelationTask` object that represents a DSP task that needs to be performed on a chunk of RF. If the track set is empty, the code obviously decides to search for tag transmissions, and if the search set is empty, the code decides to process the earliest unprocessed transmission. In these cases, when either set is empty, the code also zeros two variables that count how much processing time was spent on searching and on tracking, since that history has no use when we only need to search or track.

If neither set is empty and the code needs to both search and track, it compares the two variables that count processing time in the two modes and chooses to do the task on which less time was spent since counting started (the nearest time in the past in which both sets were not empty). When a task ends, the code updates the appropriate variable. When the scheduler decides between searching and tracking, it also considers whether the two required chunks of RF (one for searching and one for tracking) are in the future or in the past. If one is in the future, it always selects the other, to avoid blocking waiting for RF when the RF required for another task is available. The notion of future in these decisions is not with respect to real time, but with respect to the newest RF available in the radio server’s buffer; to find out that time, the scheduler requests the newest sample from the buffer and uses its time stamp as a reference for the present time.

This strategy ensures that the base station continues to search for tags even when it cannot process all the transmissions in the track set, and that it continues to track even when there are many tags to search for. Even if there are too many tags to search for and track, the base station will continue to both search and track. It will spend about half its processing power tracking tags with a known phase (but it might fail to process some of their transmissions), and it will continue to search for tags. The intent is that overloaded base stations will discover tag phases more slowly than unloaded base stations and that they will fail to process some known-phase transmissions, but that they will still attempt to localize all the tags. We have not yet explored the how to schedule searching and tracking under overload conditions and the code currently does this in a naive way.
Once a tentative scheduling decision is made, the code attempts to retrieve the necessary chunk of RF from the radio server. If that chunk is still available in the radio server’s buffer, it is sent over to the base station code and the scheduler returns a CorrelationTask object. If the chunk is no longer in the radio server’s buffer, the scheduler reports the failure and restarts the decision-making algorithm.

If the decision is to track, the length of the RF chunk in the correlation task is determined by the length of the transmission, padded with margins on both sides. The margins have two purposes. The first is to prime the FIR filters that the DSP uses (this is required only on the starting side of the expected transmission time), and the second to account for inaccuracy in the transmission time of the tag. Our experiments show that tags normally transmit within at most 30µs of the time they are supposed to, but we keep the margins longer since going much below 1ms would not reduce the cost of the DSP much, given that transmissions are several milliseconds long. This also helps avoid problems caused by the effect of temperature on the tags’ real time clocks. We can also start with much longer margins when the base station searches for a tag, to allow for tags with inaccurate clocks.

If the scheduler decides to search, it retrieves a chunk of RF that is several times longer than typical transmissions (currently a 100ms chunk), and which includes a full transmission before the previously searched chunk ends. This overlap of at least one transmission is designed to ensure that a given transmission will be fully contained in at least one searching period that overlaps it, to ensure that the correlation value will be as high as possible. Using durations that are several times longer than this overlap ensures that the average times an RF sample is processed as part of an RF chunk in searching tasks is close to 1, not close to 2. Making search chunks longer does make the per-sample cost of searching higher, because the complexity of processing $n$ samples is $\Theta(n \log n)\text{fft}$ [2], which is superlinear, but because the superlinear factor is only logarithmic, the increased cost is not significant.

If the samples requested for a searching task are no longer available, the algorithm moves the search time to the newest samples available in the radio server’s buffer. Normally, the DSP required for searching (unlike tracking) takes longer
than real time, so these overruns happen regularly. This means that the base station searches a continuous range of samples (in several different requests), then gets too far behind, moves forward in time and searches another continuous range of samples, and so on.

### 4.6 Digital Signal Processing and Buffer Management

Both the DSP and the communication with the radio server are performed by C code that the Java code calls through a Java Native Interface (JNI). RF samples and vectors derived from them remain in buffers allocated by the C code, to avoid the overhead of transporting large arrays between C and Java.

![Digital signal processing in ATLAS base stations.](image)

The overall signal flow of the DSP is shown in Figure 4.2. The first few stages
of processing take a vector of complex RF samples and produces a vector that represents a soft-detected binary bit sequence. The values in this sequence vary between $+1$ and $-1$, where $+1$ represents one symbol and $-1$ the other. In between values represent uncertainty as to whether the received sample is part of one symbol or the other. These stages are referred to as an FSK demodulator. They start with mixing the signal to the center of the baseband (zero intermediate frequency). The signal is then bandpass filtered to remove energy outside the transmitted spectrum, and then through two bandpass matched filters, one corresponding to a bit period of the $+1$ symbol and the other to a bit period of the $-1$ symbol. The output of these two filter indicates the amounts of energy in the two frequencies that represent the two symbols. The code then computes the absolute values of the outputs of these two filters, subtracts the absolute value of the $-1$ (F0) filter from that of the $+1$ (F1) filter, and normalizes the difference by the sum of the absolute values. This method results in a $+1$ if the incoming signal contains energy only in frequency F0 and in a $-1$ if the signal contains energy only in frequency F2, and in values between $-1$ and $+1$ in all other cases.

Both filtering stages are done in the frequency domain, to reduce computational cost. The Fourier transform of the composition of the filters is precomputed and stored as part of an FSK demodulator data structure in the first time that they are used. The main costs involved in applying a demodulator are the FFT of the incoming RF samples and the two inverse FFTs of the two matched-filtered signals. All FFTs are computed using FFTW, a library of high-performance FFT implementations. All the FFTs are performed on vector lengths for which a good FFT algorithm exists in the library (a product of powers of 2, 3, and 5); lengths of RF chunks retrieved from the radio server are always padded to a good FFT size.

Calling FFTW to perform an FFT task involves two stages. In the first stage the DSP code calls a planning routine and gives it the addresses of the input and output vectors, their length, and the type of FFT computation to be performed (forward or inverse, real or complex, etc.). This routine returns an FFT engine optimized for the particular task. The planning takes into account even the alignment of vectors in memory, so its output is only applicable to the specific addresses passed to the planner. In the second stage, the DSP code executes the plan, which causes FFT to actually compute an FFT. Because each plan is specific
to certain addresses and vector lengths, ATLAS allocates vectors as part of the FSK demodulator data structure and plans the FFTs on them.

When the base station needs to perform FSK demodulation, it checks whether it has already created a demodulator for the particular combination of receiver center frequency and sample rate, transmitter center frequency and deviation, and transmitter bit rate. If it has, it copies the RF samples to that demodulator and executes it. If not, it retrieves an appropriate bandpass FIR filter (these are pre-computed by Matlab code that generates Java code that represents the filters) and constructs a new FSK filter. The construction allocates space for vectors, plans the forward and inverse FFTs, and computes the FFTs of the composed filters.

The sampled soft-detected approximation of the transmitted bit sequence is then correlated with the sampled bit sequence of a particular transmission of a tag. The correlation again is done in the frequency domain. To reduce work, the FFTs of the sampled bit sequence of the transmission is precomputed and stored. To perform one correlation, ATLAS first checks if the FFT of the bit sequence at that particular signal length exists, and if it does, it invokes the correlator on it and on the output of the FSK detector. If the FFT of the bit sequence does not exist at the required length, ATLAS computes the FFT and stores it for later. In the first time that this is done for a particular vector length, ATLAS also needs to allocate space for FFT inputs and outputs and to plan the FFT.

When the correlation task tracks a tag with a known phase, the base station invokes a single FSK detector on the RF samples and correlates the FSK output with one sampled bit sequence, the one that correspond to the transmission that the base station expects to detect. But when the correlation task searches for transmissions of tags with unknown phase, the base station may invoke several FSK detectors on a chunk of RF samples, and it may correlate the output of each detector with many sampled bit sequences of transmissions at the same center frequency, deviation, and bit rate.

The C code is harder to modify than the Java code (and modifications of C code are more prone to errors), so we decided to perform the final stages of the correlator in Java. These stages include the arrival-time estimation (estimating the peak time of the correlation signal) and the estimation of the signal-to-noise ratio (SNR) of the correlation signal. In order to avoid passing a long correlation signal
The summary data structure allows our Java code to interpolate correlation values around the maximum value to estimate the sub-sample timing of the correlation peak, to compute the SNR of the correlation signal, and to determine the sign of the correlation peak, which we use to distinguish between signal and noise and. The sign of the peak can also be used to transport small amounts of data, although we do not currently use this capability.

4.7 Using the Sign of Correlation Peaks

The correlation signal that base stations compute is real (it is an inner product between two real signals, the bit sequence and the real soft-detection output of the FSK demodulator). If the RF contains the FSK signal that corresponds to the bit sequence that we correlate with, the peak (maximum in absolute value) of the correlation signal will be positive. If the RF does not contain that signal, the peak value can be either positive or negative. We refer to the sign of the correlation peak as the symbol value of the correlation task. The probability of positive and negative symbols is equal if we assume that the RF contains only Gaussian noise. If the RF contains interfering signals that are not Gaussian, the probability of positive and negative symbols depends on the bit pattern. In a long random bit pattern with an equal number of zeros and ones, the two probabilities will be roughly equal. If the pattern contains more zeros than ones or vice versa, long-lasting narrow-band interference near one of the two FSK frequencies biases the symbol value.

We use the sign of the peak in two ways. One is to determine the SNR threshold of detections of actual transmissions. We count for each tag the number of positive and negative symbols in each range of SNRs (we use 1dB ranges). In ranges of high SNRs that correspond to detections of actual transmissions, we ex-
pect to see only positive symbols. In ranges of low SNRs that sometimes result from a correlation computation on RF that does not contain the transmission, we expect to see both symbols in roughly equal numbers (at least if the number of zeros and ones in the transmission is the same). These symbol counts helped us determine the SNR threshold that we use to decide whether a correlation peak indicates the reception of a transmission or of noise.

We can also use the symbol values to transport small amounts of information from tags to the ATLAS base stations and server. If a tag inverts the two FSK frequencies, the correlation peak becomes negative. The tag can encode information in this way into its transmissions. Doing so requires inserting framing information and error detection coding into the symbol sequence, to allow ATLAS base stations to determine frame boundaries and to compensate for missing or erroneous detections. We have built some of the necessary software infrastructure required to extract data from the symbol streams, but we currently do not use this mechanism.

4.8 Remote Maintenance of Unattended Base-Stations

Base stations are scattered around the area of interest, the Hula Valley, which in our case is a rural area 175km north of our lab in Tel Aviv. Base stations need to work around the clock, unattended. Some of them are placed in cabinets in the middle of nature and accessing them is a complicated task. They send information back to the ATLAS server, but we also need to access them in order to configure them and to diagnose and resolve software and operational problem.

Base stations run Linux and we configure their pre-operating-system software (BIOS or UEFI) to restart automatically after power failures. All the ATLAS-specific programs and services start up automatically. Each program or service is run by a shell script that runs the program in an infinite loop, to ensure that it is restarted if it crashes. We normally insert a 10 seconds sleep time after each invocation of a problem, to limit program startup overhead in cases where repeated failures are caused by some unmet condition (e.g., the program depends on another program that is not running). The scripts, in turn, are all run by the /etc/rc.local script, which Ubuntu Linux runs after at the end of the boot pro-
cess. Our infinite-loop scripts are run using the `start-stop-daemon` mechanism which ensures that the script is run only once and that it runs under a user account, not under the root account.

Some base stations are connected to the Internet through local-area networks (LANs) in research facilities or rural communities. Others are connected using cellular modems (getting the cellular network connection to start automatically in Ubuntu Linux is not trivial, unfortunately). The remote connection to the stations is described in 3.2
5 Localization process

5.1 TOA Algorithm Overview

Consider the setting shown in Figure 5.1, in which we have four receiving base stations in known locations $A, B, C, D$ and one tag in the unknown location $X$ (we will explain the function of the beacon at $Y$ later). We assume that the transmitters and receivers line on a plane, but the techniques extend easily to three dimensions. Suppose that the tag transmits a signal at some arbitrary time $t_0$. The transmission reaches the receiving antenna at $A$ at time $t_0 + \frac{\|X - A\|}{c}$, where $\|X - A\|$ is the distance between $A$ and $X$ and $c$ is the speed of light. The receiver at $A$ samples the signal a little later, at $t_A = t_0 + \frac{\|X - A\|}{c} + d_A$, where $d_A$ is the analog delay between the antenna and the analog-to-digital converter (ADC). Let us assume for now that the receiver can measure $t_A$ accurately, that the same is true for the three other receivers, and that their analog delays are all the same, $d_A = d_B = d_C = d_D = d$ (these assumptions are not necessarily true but they will make the discussion simpler for now).

If we subtract the receiving time of one base station from that of another and scale the difference by $c$, we obtain the distance difference. For example,

$$t_{AB} = t_A - t_B = t_0 + \frac{\|X - A\|}{c} + d_A - t_0 - \frac{\|X - B\|}{c} - d_B$$

$$= \frac{\|X - A\| - \|X - B\|}{c}$$

or $\|X - A\| - \|X - B\| = c(t_A - t_B)$. This single equation does not fully define $X$, which, being a point in the plane, has two parameters. But the equation does
constrain $X$ to a hyperbola $H_{AB}$, since a hyperbola is the locus of points where
the absolute value of the difference of the distances to the two points is constant.
The sign of the difference tells us on which of the two disconnected curves of
the hyperbola $X$ lies. If we repeat the process for $t_{AC}$, say, we obtain another
hyperbola $H_{AC}$ on which $X$ must lie, and in theory we can now determine $X$ by
computing the intersection point of $H_{AB}$ and $H_{AC}$. Furthermore, $X$ must also lie
on the hyperbolas $H_{BC}$, $H_{AD}$, $H_{BD}$, and $H_{CD}$, which all must intersect at one point
$X$.

In practice, the receivers cannot determine $t_A, \ldots, t_D$ exactly. They only esti-
mate the arrival times, and these estimates are only approximate because of vari-
ous sources of noise; we discuss these sources of noise in 5.4. This implies that the
hyperbolas intersect pairwise but they do not intersect all at a single point, so an
algorithm must estimate $X$ from the hyperbolas or from the pairwise intersections.

The analog time delays $d_A, \ldots, d_D$ are in general not identical. The differ-
ences may be due to the use of different models of receivers, variations between
receivers of the same model, and variations in the length of the transmission line
between the antenna and the receivers. The last source of variation can be easily
accounted for, but the others are more difficult to estimate. There are at least two
ways to address this difficulty. The first is to use a beacon with a known position
$Y$ to estimate delay differences. For example, for $A$ and $B$ we have

$$t_{AB} = t_A - t_B = t_0 + \frac{\|Y - A\|}{c} + d_A - t_0 - \frac{\|Y - B\|}{c} - d_B = \frac{\|Y - A\| - \|Y - B\|}{c} + (d_A - d_B).$$

If $Y$ is known, then the only unknown in this equation is $d_A - d_B$, which we can
therefore determine (or estimate from multiple measurements). The difference is
all we need in order to generate the hyperbola $H_{AB}$, since $\|X - A\| - \|X - B\| =
c(t_A - t_B - (d_A - d_B))$.

The other way to address the unknown analog delays also solves another dif-
ficulty: unknown offsets in the clocks of the different base stations. It is hard to
ensure that all the base stations have synchronized clocks. It is easier to ensure
that they have clocks that run at almost the same rate, but with some non-trivial
offset errors. That is, when the time is \( t \), the clock at base station \( A \) shows \( t + o_A \), the clock at \( B \) shows \( t + o_B \) with \( o_A \neq o_B \), and so on. We can address this issue using the beacon at \( Y \). Suppose that the beacon at \( Y \) emits a signal at time \( t_0 \) and that the tag at \( X \) emits a signal at \( t_1 \). Station \( A \) detects the samples of the beacon at local time \( t_A^0 = t_0 + \|Y - A\|/c + d_A + o_A \) and \( B \) detects the same samples at its local time \( t_B^0 = t_0 + \|Y - A\|/c + d_B + o_B \). They also detect the tag’s signal at local times \( t_A^1 \) and \( t_B^1 \) defined similarly. Base station \( A \) computes

\[
\begin{align*}
t_A &= t_1^A - t_0^A \\
&= (t_1 + \|X - A\|/c + d_A + o_A) - (t_0 + \|Y - A\|/c + d_A + o_A) \\
&= (t_1 - t_0) + \|X - A\|/c - \|Y - A\|/c.
\end{align*}
\]

Both \( d_A \) and \( o_A \), which are not known, do not show up in this difference. The expression \( \|Y - A\|/c \) is known. If \( B \) computes a similar difference and we subtract these differences, we obtain

\[
\begin{align*}
t_{AB} &= t_A - t_B \\
&= \left( (t_1 - t_0) + \frac{\|X - A\|}{c} - \frac{\|Y - A\|}{c} \right) - \left( (t_1 - t_0) + \frac{\|X - B\|}{c} - \frac{\|Y - B\|}{c} \right) \\
&= \frac{\|X - A\| - \|X - B\|}{c} - \frac{\|Y - A\| - \|Y - B\|}{c},
\end{align*}
\]

or

\[
\|X - A\| - \|X - B\| = c \left( t_A - t_B \right) + \frac{\|Y - A\| - \|Y - B\|}{c},
\]

which defines a hyperbola (all the quantities on the right are known).

We note that the RF samples of multiple receivers at known positions can be used to directly estimate the location of tags [17]; it is not strictly necessary to estimate signal arrival times first and from them to estimate locations. However, the direct position determination method [18, 17] requires that all the RF samples of transmissions that need to be localized be available to processing together in one computer, which increases the bandwidth required between receiving base stations and the server that runs the localization algorithm.
5.2 Estimating the Arrival Time of Known Signals

In ATLAS, tags and beacons transmit signals that are known to receiving base stations. Let the signal that a tag (or beacon) needs to transmit be \( s(t) \). We assume that \( s(t) = 0 \) for \( t < t_0 \) and for \( t > t_0 + d \), where \( d \) is the duration of the signal. What the tag actually transmits is \( \tilde{s}(t) \), where \( \tilde{s} \) differs from \( s \) slightly because of the tag’s inability to reproduce \( s \) exactly (in some tag designs the difference may be very small, but there is always some difference). The signal that the receiver actually receives is \( a\tilde{s}(t) + n(t) \), where \( n \) represents noise and/or interference and \( a \) is the attenuation factor of the path between the transmitter and the receiver. The receiver analog front end amplifies and filters the signal, which introduces additional noise and distortion. The receiver then samples the processed incoming signal, which we denote by \( \bar{s} = a_1\tilde{s} + n_1 = a_2s + n_2 \) where \( a_{1,2} \) are scalar amplification factors and \( n_{1,2} \) represent noise and distortion.

To estimate \( t_0 \), the receiver computes the discrete cross correlation signal

\[
\text{xcorr}_{s,\tilde{s}}[j] = \sum_{i=\infty}^{\infty} s[i]^{*}\tilde{s}[i+j]
\]

where \( s[k] \) and \( \tilde{s}[k] \) represent the \( k \)th sample of a signal and where \( s[k]^{*} \) represents the complex conjugate of \( s[k] \). This discrete correlation signal can be interpolated to approximate the continuous correlation signal

\[
\text{xcorr}_{s,\tilde{s}}(t) = \int_{\infty}^{\infty} s(r)^{*}\tilde{s}(r+t)dr
\]

(assuming, of course, that the sampling rate is high enough relative to the frequency content of \( \tilde{s} \).)

If \( s \) is a modulation of a long random or pseudo-random sequence and if \( \tilde{s} \) is close to a multiple of \( s \), then \( \text{xcorr}_{s,\tilde{s}}(t) \) has a distinct peak at \( t_0 \). ATLAS estimates this peak and reports it location as the arrival time estimate of \( s \).

Fundamentally, errors in arrival-time estimates depend on three factors. One is multipath propagation, which causes the transmitted signal to arrive at the receiver several times over different paths. The other is the total noise \( n_2 \), which consists of \( s - \tilde{s} \) (the difference between what the transmitter needs to transmit and what
of noise and interference that the receiver receives at its antenna, and of noise and distortion that the receiver itself generates. The third is the distinctiveness and sharpness of the correlation peak, which depends mainly on the time-bandwidth product of $s$. Signals that have long duration, high bandwidth, and good signal-to-noise ratios (SNRs) have more accurate arrival time estimates than short, narrowband, and weak signals. Under certain assumptions, the Cramer-Rao bound (CRB) bounds the variance of arrival-time estimators as a function of SNR and the time-bandwidth product. We omit technical discussions of this and more refined bounds from this discussion; it is sufficient to know that these three factors (SNR, signal duration, and signal bandwidth) are critical for accurate arrival time estimation.

5.3 Solving the Location Equations

Given a transmission at time $t$ from location $p = [x \ y \ z]$, the estimated reception times at $m$ base stations at positions $p_i = [x_i \ y_i \ z_i]$ are

$$t_i = t + \frac{1}{c} \| p_i - p \|_2 + n_i = t + \frac{1}{c} \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} + \tilde{n}_i$$

or

$$(t_i - t) - \frac{1}{c} \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} = \tilde{n}_i$$

where $c$ is the propagation speed and the residuals $\tilde{n}_i$ represents measurement and estimation noise. Given $m$ such measurements, we estimate $t$ and $p$ by minimizing $\sum_{i=1}^{m} \tilde{n}_i^2$. We denote by $d_i$ the (unknown) distance from $p_i$ to $p$, $d_i = \| p_i - p \|$.

We can scale the residual by $c$ without changing the optimal $t$ and $d$. This has the effect of changing the units of the residuals from time to distance, which is probably more useful to users. We denote the distance residuals by $n_i$,

$$c (t_i - t) - \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} = n_i.$$
We measure the overall residual of a solution using

\[ r = \frac{\|n\|_2}{\sqrt{m}} = \frac{1}{\sqrt{m}} \sqrt{\sum_{i=1}^{m} n_i^2} \]

which is independent of \( m \) in the sense that if every \( n_i = n_0 \) then \( r = n_0 \).

### 5.3.1 Shifting and Scaling

If we shift all the \( t_i \)s by some constant \( t_0 \), we get the same residual values for \( t_i \)s that are also shifted by \( t_0 \), since the residual only depends on differences \( t_i - t \). By shifting we can avoid large \( t_i \)s and large \( t \)s that can adversely affect some minimization algorithms. Reasonable ways to shift are to subtract from all the \( t_i \)s their average or median.

We also use nanoseconds rather than seconds as the time units. This scales \( c \) from 299792458 to 0.299792458 which makes \( t_i - t \) similar in size to \( d_i \), rather than eight or nine orders of magnitude larger. Some algorithms are totally insensitive to this, but some are.

### 5.3.2 Eliminating the Time Unknown

It is easy to find the time \( t^* \) that minimizes the sum of squares for a given \( p \) because \( t^* \) is the solution of a linear least squares problem with one variable,

\[
 t^* = \arg \min \frac{1}{c} \left\| \begin{bmatrix} -c \\ -c \\ \vdots \\ -c \end{bmatrix} \begin{bmatrix} t - \begin{bmatrix} -ct_1 + d_1 \\ -ct_2 + d_2 \\ \vdots \\ -ct_m + d_m \end{bmatrix} \end{bmatrix}_2. \]

The QR factorization of the coefficient matrix is

\[
\begin{bmatrix} -c \\ -c \\ \vdots \\ -c \end{bmatrix} = \begin{bmatrix} 1/\sqrt{m} \\ 1/\sqrt{m} \\ \vdots \\ 1/\sqrt{m} \end{bmatrix} (\begin{bmatrix} -c \sqrt{m} \end{bmatrix})
\]
so the $t^*$ is

$$
t^* = (-c \sqrt{m})^{-1} \begin{bmatrix}
1/\sqrt{m} \\
1/\sqrt{m} \\
\vdots \\
1/\sqrt{m}
\end{bmatrix}^T \begin{bmatrix}
-ct_1 + d_1 \\
-ct_2 + d_2 \\
\vdots \\
-ct_m + d_m
\end{bmatrix} = -\frac{1}{cm} \sum_{i=1}^{m} ct_i + d_i = \frac{1}{cm} \sum_{i=1}^{m} ct_i - d_i.
$$

This gives us a new expression for $n_i$,

$$
n_i = ct_i - \frac{1}{m} \sum_{j=1}^{m} (ct_j - d_j) - d_i.
$$

Some optimization algorithms require derivative information. We give here the partial derivatives of $n_i$. We have

$$
\frac{\partial d_i}{\partial x} = \frac{2x - 2x_i}{2d_i} = \frac{x - x_i}{d_i}.
$$

so

$$
\frac{\partial n_i}{\partial x} = -\frac{1}{m} \sum_{j=1}^{m} \frac{\partial d_j}{\partial x} - \frac{\partial d_i}{\partial x} = -\frac{1}{m} \sum_{j=1}^{m} \frac{x - x_j}{d_j} - \frac{x - x_i}{d_i}.
$$

The expressions for partial derivatives with respect to $y$ and $z$ are similar.

5.3.3 Linearized Expressions

If $m = 3$ and $z$ is assumed to be known (or if $m = 4$ and $z$ is unknown), the number of constraints and the number of unknown variables is the same, so we should be able to find $t$ and $p$ such that all the $n_i$s are zero. We can simplify the resulting equations by multiplying

$$
c(t_i - t) - d_i = 0
$$

by $c(t_i - t) + d_i$ to obtain

$$
c^2 (t_i - t)^2 - d_i^2 = 0
$$
or

\[ c^2 (t_i^2 - 2t_i t + t^2) - (x_i^2 - 2x_i x + x^2) - (y_i^2 - 2y_i y + y^2) - (z_i^2 - 2z_i z + z^2) = 0. \]

We refer to \( c^2 (t_i - t)^2 - d_i^2 \) as quadratic residuals. If we denote

\[ w_i = \begin{bmatrix} -2c^2 t_i & 2x_i & 2y_i & 2z_i & c^2 & -1 & -1 & -1 \end{bmatrix} \]

and

\[ b_i = -c^2 t_i^2 + x_i^2 + y_i^2 + z_i^2 \]

we can express the quadratic residuals as \( w_i \begin{bmatrix} t & x & y & z & t^2 & x^2 & y^2 & z^2 \end{bmatrix}^T - b_i \)

and the system of equations that set them to zero as \( W \begin{bmatrix} t & x & y & z & t^2 & x^2 & y^2 & z^2 \end{bmatrix}^T = b \). We denote \( u = \begin{bmatrix} t & x & y & z & t^2 & x^2 & y^2 & z^2 \end{bmatrix}^T \) to simplify the notation.

Linear combinations of \( w_i u \) in which the coefficients sum to 0 contain only constant terms and terms that are linear in \( t, x, y, \) and \( z \); the quadratic terms cancel out because their coefficients are 1 or \(-1/c\). For example,

\[
 w_1 - w_2 = 2 \begin{bmatrix} c^2 (-t_1 + t_2) & x_1 - x_2 & y_1 - y_2 & z_1 - z_2 & 0 & 0 & 0 & 0 \end{bmatrix}.
\]

Therefore, if \( L \) is a matrix with zero row sums, \( L \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T = 0 \), then \( LWu = Lb \) is a linear system of equations in \( x, y, z, \) and \( t \) only; the quadratic terms drop out because the last four columns of \( LW \) are zero. Unfortunately, \( L \) is clearly singular, so solving \( LWu = Lb \) for \( t, x, y, \) and \( z \) does not imply that \( Wu = b \). It only implies that either \( Wu = b \) or that \( Wu - b \) is in the null space of \( L \). If \( L \) is rank deficient by exactly one then multiples of the constant vector are its only null vectors, which implies that

\[ w_i u - b_i = w_j u - b_j \]

for all \( i \) and \( j \) (this equation holds whether \( Wu = b \) or not). That is, by multiplying the quadratic-residual equations by \( L \) we try to force the quadratic residuals to be equal to each other, not necessarily to be zero.

We need to ensure that \( L \) has only one null vector, the constant vector, otherwise \( LWu = Lb \) would not imply anything useful. This means that the number of
rows in $L$ needs to be $m - 1$ or larger (the number of columns is always $m$). We can use $L$s of the form

$$
L = \begin{bmatrix}
-1 & 1 & 0 & 0 \\
-1 & 0 & 1 & 0 \\
-1 & 0 & 0 & 1
\end{bmatrix},
$$

or we can take a random $k$-by-$m$ matrix $L$ with zero row sums for $k \geq m - 1$, or a matrix with all the possible rows with one 1 and one $-1$. The rank of $LW$ will usually be $\min(4, m - 1)$ or $\min(3, m - 1)$ if $z$ is known; it cannot be larger than 4 (or 3) because $LW$ has only 4 (3) nonzero columns, and it cannot be larger than $m - 1$ because $LW$ has only $m - 1$ rows. If $\text{rank}(W) < \min(4, m - 1)$ then the rank of $LW$ will also be necessarily smaller, but this should happen very rarely.

If the rank of $LW$ is 4 (or 3 if we do not need to estimate $z$), the linear system $LWu = Lb$ determines $t$, $x$, $y$, and $z$ uniquely; the solution is the one that makes the quadratic residuals as equal as possible in the least squares sense with respect to the given $L$.

If $m = 3$ and we need to estimate $z$ or if $m = 2$ and we do not, $LW$ will be rank deficient by one (ignoring the zero columns that correspond to squares of $t$, $x$, $y$, and $z$). This means that there is a one-dimensional space of solutions of the form $u = u_b + \alpha u_n$ where $u_b$ is some solution of $LWu_b = Lb$ and $u_n$ is a vector from the one-dimensional null space of $LW$, $LWu_n = 0$ (we ignore the zero columns of $LW$ when constructing a basis for the null space). For any such $u_b + \alpha u_n$ the quadratic residuals are equal; we can now search for an $\alpha$ such that they are zero. We set up one such equation

$$
w_i(u_b + \alpha u_n) - b_i = 0
$$

or

$$
c^2(t_i^2 - 2it + t^2) - (x_i^2 - 2x_i x + x^2) - (y_i^2 - 2y_i y + y^2) - (z_i^2 - 2z_iz + z^2) = 0
$$
for $t = t_b + \alpha t_n$ etc., or

\[
(c^2 t_n^2 - x_n^2 - y_n^2 - z_n^2) \alpha^2 \\
+ 2 \left( -c^2 (t_i t_n - t_b t_n) + x_i x_n - x_b x_n + y_i y_n - y_b y_n + z_i z_n - z_b z_n \right) \alpha \\
+ c^2 \left( t_i^2 - 2 t_i t_b + t_b^2 \right) - \left( x_i^2 - 2 x_i x_b + x_b^2 \right) - \left( y_i^2 - 2 y_i y_b + y_b^2 \right) - \left( z_i^2 - 2 z_i z_b + z_b^2 \right) = 0.
\]

We can solve the equation for $\alpha$ and set $u = u_b + \alpha u_n$ to obtain solutions for which the quadratic residuals are all zero. If there is a solution for which the original residuals are all zero, at least one of the $\alpha$s will be real and will correspond to this exact solution. If there is no exact solution that zeros the original residuals, there may still be one or two solutions that zero the quadratic residuals (in these solutions, some of the propagation times will be negative), or there may be no real solution to the quadratic equation above.

If $m$ is larger and we solve $L Wu = Lb$ in the least-squares sense, the squared residuals will be as equal as possible (in the least-squares sense for a given $L$), but at least in theory it is possible that some of the propagation times will be negative. We add inequality constraints that force the propagation times to be nonnegative, but this is probably too expensive.

### 5.4 Source of Errors in TOA Localization

Location estimates in TOA (Times Of Arrival) systems contain errors from several sources. Understanding these sources of error is important for understanding how system-design decisions affect localization accuracy. The sources of errors are:

1. Errors in the specification of the location of the antennas of receiving stations and beacons. We assumed that these locations are known, and any error in their localization translates into an error in the localization estimates of tags. These locations need to be determined accurately enough using established surveying methods.

2. Errors in arrival-time estimates in receiving base stations, an issue that we covered in the previous section.
3. The other time-measurement issue that arises in Equation (5.1) is the accuracy of the clock that the base station uses to determine the difference between $t_A = t_A^A - t_A^B$. In ATLAS, we use the radio receiver’s sampling clock to determine this difference. If that clock is a bit too fast or a bit too slow, the difference will contain a corresponding error, even if each arrival time estimation is completely accurate with respect to the local clock’s state at arrival time. In ATLAS we use two techniques to reduce this error. One technique is the use of a stable and accurate clock source for the sampling clock of the radio. More specifically, we use a GPS disciplined oscillator (GPSDO) as a reference clock for all the radio’s oscillators, including its sampling clock. The other technique is to have the beacons transmit frequently (about every second), so that $t_A$ is small in absolute value. If the sampling clock’s frequency is accurate to within a relative error of $10^{-9}$ (typical for good GPSDOs) and $t_A \approx 1s$, then the absolute error in $t_A$ is on the order of 1ns. The mean relative frequency error of GPSDOs becomes smaller as the measurement period (say $10^{-9}$ over a second but only $10^{-12}$ over 24h), but the improvement is sublinear, so shorter $t_A$’s lead to smaller absolute errors.

4. The sensitivity of the maximum-likelihood location to time-difference errors and to errors in the location of fixed stations. This sensitivity depends on the geometry of the intersecting hyperbolae (or hyperboloids). Intersections at sharp angles and/or in areas where the hyperbolae have a high curvatures are highly sensitive to measurement errors. Intersections at angles close to 90 degrees in areas where the curvature is low are relatively insensitive. The overall sensitivity depends on these geometric relationships between all the hyperbolae/hyperboloids that define the location of the unknown transmitter, and it is known as the Geometric Dilution of Precision (GDOP).

5.5 Coordinate System

The coordinate system used in ATLAS is ITM, Israeli Transverse Mercator. It is a Cartesian coordinate system, meaning that the surface coordinate are expressed
in X and Y values, which correspond to meters east and north of the coordinate system origin. We also regard Z, the height value, so a location in ATLAS is a three dimensional coordinate \((x, y, z)\), with the unit being meters.

The coordinate system is expressed in ATLAS by the fact that the coordinates of the base stations and beacon tags, which are constant values in the program, are expressed in ITM coordinates. The localization algorithm bases its solution in the coordinates of these values.

The locations of the base stations and beacon tags were measured by GPS devices, which use spherical coordinates in the WGS84 system.

The ITM coordinate system is incorporated into the ATLAS system by the fact that the base stations and beacon tags coordinates are expressed in this coordinate system. The localization algorithm calculates locations of transmitters by relating them to these constant locations.

The coordinates of the base stations are taken from the GPS devices that are connected to the stations USRPs. It is not ideal since the GPS antennas are not located exactly where the RF antenna is. Theoretically it adds some constant bias to the system’s calculations. The coordinates of the beacon tags were taken by using a GPS device on the spot of the tag, so they are more accurate.
Figure 5.1: Locations of base stations, a beacon, and a tag that needs to be localized. The locations of base stations and the beacon on this map reflect approximate locations of actual ATLAS stations.
6 Main Server

The main server collects detection reports from the base stations, archives them in the database and process them to create input for the localization algorithm. It receives back the localizations and archives those as well, and also sends them to the consumer clients of the ATLAS system.

6.1 Processing Incoming Detections

This section describes the process of the main server beginning from receiving detections from the remote base stations and ending in sending locations of tags to the system’s consumers. The first processing steps are described in figure 6.1.

6.1.1 Receiving Input from Remote Stations

When a remote station connects as a client to the server’s incoming connections listener, the server initiates a new thread for the connection (which makes the number of threads in the main server process at least the number of connected stations), and allocates a context structure in memory for that specific base station. The base station context in the server is implemented as the BaseStationListener object. Basically, the only unique context for each connection is a connection to the database. As an option for future development, this context can hold variables meant for stations administration and statistics. The base stations send the server detection information using objects of the Detection class, which holds such information as detection time, tag id, correlation SNR and more. When a detection message is received, a record of the detection details is written to the database.
Every detection is saved in the database for the reason that if the localization process described in this section fails then later, an offline localization process can be done using the database instead of the online incoming detections. In case of disconnection from the base station the thread is closed and the server holds no memory of the connection. It is up to the base station to reconnect.

6.1.2 Grouping Together Different Detections of a Single Transmission

The base stations do not have any interaction with each other. It is up to the server to cluster together the different detections of the same transmission, for each transmission. In other words, for every new detection received the server needs to group it together with existing detections of the same transmission received from other stations, or to realize that it is the first detection of a transmission, and open a new group.

In order to make the process of finding a detection’s group more fast, the groups in memory are mapped first by their tag id, and for each tag id they are mapped sorted by their time. Usually, when the system is not in some error recovery process, every arriving detection actually belongs to the latest transmission happened and so its group will be the first one in its tag id’s queue so it will not have to iterate the queue much.

A detection is added to a group of detections (the class TxDetSet) if its reported detection time is different than the time of the other detections in the group no more than a few milliseconds. This condition is suitable counting on the fact that a specific transmission is emitted only once a second or more and that the distance between the receivers is a few kilometers. A transmission travels one kilometer in about 3 microseconds, so the time differences between the detections can not be more than a few microseconds. Considering that there will not be two different transmissions during such a short period, then any two detections of the same transmission within a few milliseconds have to belong to the same transmission.
6.1.3 State Machine of a Detection Group

Every detection group created needs to be removed from memory later on, to make room for new detections. There are a number of decisions that need to be made in a detection group work cycle. Every incoming detection triggers a transition in this state machine. The different states are described next:

1. **A detection has no group: It is new:** When a received detection is newer than the oldest group in memory, and does not belong to any group, a new group is created with it as its first member. If the number of existing groups has reached the allowed capacity in memory, the oldest group will be removed, without localizing it. In addition, if the detection is of a beacon tag, a reference to the new group will be added to a queue of beacon detections.

2. **A detection is the 2nd in the group:** It is added to the group.

3. **A detection is the 3rd in the group:** It is added to the group and a timer is set for a localization process to start 30 seconds later.

4. **A detection later than the 3rd arrives before the timer expires:** It is added so it would be included in the localization.

5. **The localization timer expires:** The 3 or more detections are processed into a LocalizationTask object, as described later on in this section. The group is removed from memory.

6. **A detection arrives after its group’s localization process has started:** The group was actually removed from memory and so no group will be found for this detection. If its later than the oldest group in memory, it will be treated as a new transmission and will open a new group. If it is older than the oldest group, it would just be ignored. In both cases the detection will not be a part of the localization process and so the update of the incomplete localization will have to be done later, based on the database record of that detection.
Unless there is currently a recovery from a disconnection in the system, state 6 shouldn’t occur and the 30 seconds of the timer (set in state 3) are enough until the last detection is received in the server and added to the group. The case of late detections happens when a station recovers after a disconnection of some time and the detections that the station saved during the disconnection are sent to the server. Those detections will be older than the detections coming in from the other stations and so they will be ignored.

6.1.4 Matching a Beacon Transmission

After a localization timer has expired, a group of 3 or more detections is being prepared for localization. First, there needs to be found a detection group of a beacon transmission, a bit earlier than the localized transmission, because we need to localize using times which are related to some beacon and not the actual times of the detections. The conditions for matching a beacon transmission to a localized transmission:

- **Tag and beacon transmissions have same frequency:** The RF filters in the receivers’ antennas have the effect of frequency dependent time delay. It means a signal in one frequency will pass through the filter faster than a signal in a different frequency. This delay is insignificant when doing calculations based on transmissions with the same frequency, since we subtract times one from another.

- **Beacon and tag detections come from same stations:** The group of stations of the beacon group must contain the group of station of the localized transmission. If a detection of the localized transmission will not have a beacon detection from the same station, it will not be able to take part in the localization.

- **The beacon transmission and localized transmission times are close:** The time difference is at most 10 seconds between the two transmissions. It is described in detail in section (9.1) that there are clock errors in the receivers. For that reason we want the beacon time to be as close in time to the transmission time, so the clock errors won’t sum up too much. It
is possible however, to do the localization calculation using negative times - meaning that the matched beacon transmission is later than the localized transmission.

The beacon detection groups in memory are regular detection groups. They are, however, referenced from an extra beacon data structure. The beacon reference structure maps beacons by their frequency (Each tag transmits different transmissions. Transmission IDs can be translated to frequency values), and for each frequency there is a circular queue of references to beacon detection groups of that frequency. This structure, like the detection groups queue, is a static (global) member of the object BaseStationListener.

6.1.5 Calculating Beacon-Relative Detection Times

In this sub section the tag we are interested in localizing will be referred to as “the tag” while the beacon tag will be referred to as “the beacon”, although the beacon is an ordinary tag, only with a known constant location.

RF samples produced by the receivers are timestamped with the receiver’s clock time, in the format of Unix time: the number of seconds since the “Epoch” (January 1st, 1970). The purpose of using beacon detections is to provide a common “zero time” for the specific detection group, i.e a common point in time which is calculable by all the stations, and can be used to synchronize their detection timings. Theoretically, the GPS disciplined clocks in the receivers are supposed to be synchronized so the time stamps the receivers generate should be usable. As described in section 9.1 the clocks are not synchronized to the accuracy level of nanoseconds. They are, however, stable enough so within a time range of a few seconds they won’t drift away from each other and so can be considered synchronized. It means that although the stations’ reported time stamps are unusable, the time difference between two preceding detections is accountable, if this difference is of a few seconds.

Since we know the exact location of the beacons and the base stations, then for a given beacon detection in some station we can calculate the time this transmission was emitted from the beacon, according to this specific station’s clock,
by subtracting from the beacon detection time the propagation time between the beacon and the station. Using this calculable beacon emission time we synchronize a group of detections by calculating for each station what was that station’s time when the beacon transmission was emitted. We use each such individual time point (which should be the same in all stations, but since the clocks have errors, its not) as the zero time for each such station for the specific tag’s transmission detection, and we express the detection time as the time since that “zero time”.

The time difference, between the beacon detection and the tag detection, should be exactly the same in all stations, since the signals travel in a constant speed, and so the different distances of the receivers from the transmitters have no effect on this time difference. To summarize, in order to synchronize the different stations’ timings, we need to have an event which all stations can accurately time and so we use a beacon transmission that was detected in all of the stations.

A mathematical description of the calculation, based on an example:

• Assume the transmission of a tag which followed a transmission of a beacon. The tag transmission was detected by stations \{a, b, c\} in the times: $t_a$, $t_b$, $t_c$ and the beacon transmission was detected by stations \{a, b, c, d\} in the times: $T_a$, $T_b$, $T_c$, $T_d$. We do not care about station d since it didn’t detect the tag.

• Suppose, for station a for instance, that the distance between it and the beacon is $d_a$. We can calculate the time any RF signal travels from the beacon to station a (c is the speed of light):

$$\Delta T_a = \frac{d_a}{c}$$

• Since a’s clock reported the beacon detection in time $T_a$, then in a’s clock the beacon has transmitted in time:

$$T_{0a} = T_a - \Delta T_a$$

• We calculate the time difference, in a’s clock, between the beacon transmis-
sion emission and the localized tag transmission detection:

\[ t'_a = t_a - T_{0a} = t_a - T_a + \Delta T_a \]

After applying this calculation to all the detections, we get the new set of times: \( t'_a, t'_b, t'_c \). These times represent the time past in each station between the beacon emission and the tag detection. These are the times that we send to the localization algorithm. Notice that all information about the beacon is not needed anymore.

The beacons-to-station timings (in the example, these are \( \Delta T_a, \Delta T_b, \) and \( \Delta T_c \)) are constant and so they are calculated once. We create at server initialization time a constant data structure that maps every station/beacon pair to a time value of the travel time of a signal between them. This mapping structure and other beacon related functionality is in the Beacon object.

We have showed here that a station detection time report is based on the difference between two time stamps in that station. It means we have no need in the absolute values of the time stamps. This is why we actually use that number of samples received between that two events in order to express the time. This integer value is saved as the “sample clock” and only in the server it is converted to time units, by dividing the number of samples with the receiver sampling rate (which is known to the server).

### 6.1.6 Sending Localization Tasks to the Localization Algorithm and Receiving Back Locations

The localization algorithm takes as input for each transmission only a set of timings (of a single transmission) and the matching station IDs of the timings. The LocalizationTask object is initialized with this set of times described in the previous section and the IDs. The algorithm actually needs the stations’ locations but it fetches them using the IDs. The LocalizationTask object also holds the tag ID and the transmission ID which are not needed for performing localization but are needed later for archiving the solution in the database. Once the localization task is sent away to be localized, it is removed from the server’s memory.

The server receives back from the localization algorithm the solution, which
is the estimated location of the tag in the reported detection time. The data is wrapped in the LocalizationSolution object which has as members the location, the time of one of the detections (as a representative), the number of detecting stations (The more stations participated, the more reliable the estimation is) and the tag and transmission IDs. The server archives the solution in the database and also sends it forward to the consumer of the ATLAS system - the Hebrew University biology lab in Jerusalem.

6.1.7 Sending Locations to Consumers

The consumer of the ATLAS system is the Eco-movement group in the Hebrew University of Jerusalem. The interface between the server in Tel Aviv University and the consumer is by an IP connection in which the consumer connects as a client to the server and is streamed with location data. The server side holds a message queue (described in 3.5) which buffers output locations in case of consumer disconnection. The data is transferred using Java serialization, like the rest of the system’s data connections, as described in 3.4. This method forces the consumer side to have the same software as the server for the message objects. This is an important issue since it obliges the server’s developers to provide software to the consumers of the server. We share with the other lab a subtree of our program source tree and by that let them be involved in the development and add their own software to their side of the program.

It is, however, possible that in case that the server maintainers won’t want to share their software, the data transfer protocol won’t be Java serialization based, but rather a custom protocol that will have to be implemented by the consumer.

6.2 Database

We use a Mysql database, provided by the Tel Aviv University computer science school IT team. We have two database tables: one for the detections and one for the locations (processed from the detections). Every thread in the server that uses the database has its own database connection. It means a connection for every base station listener and one for the localization server.
6.2.1 The Detections Table

This table holds every transmission detection reported by any base station, as soon as it arrives to the server. Its main fields are the detection time, the tag and transmission IDs and the base station ID. Saving all of the detections is very important as it provides the possibility to re-run the localization process over the raw data. The above on-line localization process does not always process all of the data it receives. If some station reconnects after a failure, the server does not match the “old” detections flowing in with the detections of the same time that were received earlier from other stations. If a consumer of the system considers some point in time to be crucial (for instance, few hours of a field experiment) he should re-run the localization process over the detections table, after querying for that time period.

6.2.2 The Localizations Table

This table holds the results of the localization algorithm. A location is stored in the table when the main server receives it back from the localization server. The location is saved as a Cartesian coordinate of three dimensions: x, y, z. Other important data fields are the tag and transmission IDs, the time and the number of base stations participated. The location record of a detection can be rewritten, if a better localization has been made. For instance, If a base station reconnects after a failure the localization process can be re-run with the extra station, and update the location result (and number-of-stations field).

6.3 Simulator Program

ATLAS includes a simulator program for testing the server. It simulates a group of base stations, sending tag detection reports which are created synthetically. It was developed before there were real base stations working, in order to develop the server and every time a new server feature needs to be tested, it can be done so using this simulator. The simulator works in such a way that from the server point of view nothing is changed and no special configurations are needed.
Simulating a tag: The main thread of the simulator program simulates detections of a moving tag. It creates a geographic path of random steps in the area of the base stations, a step in each second. Each step is a random coordinate, limited in its distance from the previous step and in the change of direction angle of the previous step’s direction. The trail starting point is some point inside the area of the base stations. In order to be able to test the accuracy of the localization algorithm, each new step is saved in a file together with its time. This way the localization algorithm can compare its estimations to the real locations. See figures 9.8 and 9.9 (from the results chapter) for a visualization of such a comparison.

Simulating base stations: The simulator program takes as input a file describing a list of base station coordinates. It then opens a different server connection for each such made up base station. The ATLAS custom connection class implements a thread, so the simulator program process holds a thread for each base station connection and the main thread which creates synthetic tag detections, as described above. For each step on the made up geographical path created by the tag simulator, a detection time is calculated for each base station. The timing calculation is simple since the base stations’ coordinates are given, the “tag” current location is given, and the signal travel speed is the speed of light. Each timing value is then added with another random value of a size of a small fraction of the timing value, to simulate the inaccuracy caused by noise of all kinds in the real system. Each (simulated) base station sends the server its timing message. Each server connection is delayed in some random small time before sending the timing message so the messages won’t be sent all at once and will act more like real remote connections.

Processing the synthetic detections: The server works as usual without any modifications. The database used is different from the real one, of course. The Matlab localization program can also run in the regular way but has another special simulation mode in which it compares the real locations and the calculation results. It is described in section 9.3.
Figure 6.1: The main server collects detection reports from the base stations and groups them in memory according to their tag, transmission ID, time of arrival and whether they are beacon tags. Once a detection group has three base stations or more, a localization task is created by subtracting a matching beacon detection group.
7 System Deployment and Maintenance

The system was deployed in the Hula Valley gradually until it reached the current status in which there are five receiver stations and two stationary tags working. At first we deployed one receiver station and one stationary tag. The station was at the comfortable location of the research center in the Hula Agmon, where it was connected to the Internet via an Ethernet cable. The tag was attached to a power outlet in some small building in the Valley. Since our base station software wasn’t stable, we had the possibility to ask for assistance on the spot from the researchers at the center. In a few times we asked someone to connect to our PC with a monitor and a keyboard and we guided him by phone until connection was restored. The next level was an addition of a second base station in a Kibbutz in the area. Localization can be calculated using at least three stations, but we could conduct the clocks synchronization tests that are detailed in subsection 9.1.2. The second station was also connected to Internet inside the local area network of the Kibbutz, and there we learned that although Ethernet connection is very fast the dependence on the local area network router caused all kinds of configuration problems. We decided to use cellular modems for all of the stations. In the next step we added another 2 stations to have a total of 4 stations. Then the system could have start doing localizations. All kinds of experiments with tags attached to animals have started to be conducted. Before we figured out exactly how to tune the cellular modem reconnection script so it will be reliable enough, we had to have someone that will reach the base stations whenever a disconnection occurred. A technician was hired and in the first few months since the 4 stations were deployed he was sent now and then to reconnect base stations and fix prob-
lems. Two of the base stations were located in hard to reach places where every visit demands to be coordinated in advance. Gradually we have fixed problems and developed the reconnection script so it was robust enough and so the technician wasn’t needed anymore. Most of the problems we had, which demanded the technician visit, were Internet disconnections caused by the cellular network or PC reboots caused by temporary power downs. During the time a station was disconnected it usually continued to gather detections and queue them in the file system, so upon reconnection the missed out data was sent to the server. Figure 7.1 shows the connection status of the five station during the last year. The next step, which is the recent one, was to add a fifth base station and in addition to move the existing stations to higher positions on masts of a cellular company’s antenna sites, in order to improve detection of near-ground tags.

The process of deploying a base station has become easier from time to time. In the first two deployments Prof. Toledo and myself have accompanied the wildlife researchers in the on-site deployment. In the next two times I alone joined them. On the next deployments the system was ready to work right away when it was connected and the deployment was done only by connecting the components, without using a monitor and a keyboard.

![Figure 7.1: Work days of the five stations starting April 2014. Missing points indicate times of disconnection.](image-url)
8 System Configuration

The ATLAS system server and base stations processes are initialized with many configuration values such as IP addresses, tag IDs etc. While some of these values rarely change, such as the server IP address and port, others are changed for each experiment, such as the list of tags to currently track.

The configuration values are saved in text files, in table formats. The files are read and parsed by the program on initialization. This allows, upon change of configuration, to re run the program without changing the source code or re-compiling it again.

The configuration files, their content and their uses:

- **The base stations file**: a row for each base station. Main fields are the location coordinates, location name, the PC host name and the USRP serial number. The base station program figures on which location it is out of its host name. The main server creates a data structure of signal propagation times between each beacon to each station using the stations locations from the file. The Matlab localization algorithm uses the stations’ locations for the localization.

- **The beacons file**: for each beacon tag, lists its tag ID and location coordinates. This file is used by the server. It uses it to tell which of the tags is a beacon tag and it also uses the location to create the beacon-to-station data structure mentioned previously.

- **The tags file**: The file that lists all of the possible tags. For each tag, it describes each of its transmissions characteristics. This file is also used for
programming the tags. The server uses the transmission frequencies listed in the file to match detections to beacon transmissions. The base station program uses the tags’ duty cycle and code id for the tracking and detection processes.

- **The track file**: This file only lists for the base station program which tags from the tags file it should search for. This is done because the less tags the program will search, the more efficient it will be. Whenever a new tag is programmed and added to the stations’ area, this file should be updated in each station.

- **General config file**: A file listing the server address, database login values and other such data.

Whenever a file is updated it should be committed (checked-in) to the source control system, and checked out in the other components. It means that an update that is relevant for the base stations requires updating the config files in each station manually (via remote access of course). Since the number of stations is not planned to be more than about 10, it is a reasonable solution. A possible more sophisticated method would be to have the server broadcast an update to the other components. It will exempt the need to update the other components manually and also the need to stop and rerun the program.

For the server, another configuration feature is configuration sets. The set of configuration files described above appear in more than one directory of configuration. For instance, the configuration of the real system will have the base stations file with the real base station while the directory of the simulation program will have made up stations. The configuration directory path is given as an argument for the server process.
9 Experimental Results

The experimental results described in this chapter regard both lab and field experiments. These experiment were performed in order to analyze the system’s equipment and software. Analysis of the localization accuracy is in progress and it will not be discussed here but rather in future work.

9.1 Receivers Clock Errors Analysis

We made experiments in order to analyze the receivers’ clock errors detailed in the previous sections. The first experiment was a basic calibration test of the Time-Of-Arrival concept. We wanted to make sure that two radio receivers which are located next to each other (conceptually considered as zero distance) will output tag detection time reports with almost zero difference. During every experiment, we had one or two tags transmitting, each tag transmitting once a second. Each tag transmitted 6 different transmissions: three short ones of about 2000 bits and three long ones of about 8000 bits. According to this tag configuration each unique transmission repeated every 6 seconds. Since the tags were located in the same room there were almost no miss-detections. It means that every transmission was detected by both receivers. We wanted to test accuracy - whether the time values are the same, and stability - whether the clock frequency is the same.

9.1.1 Clocks Accuracy

In the data analysis we have paired the detections, each pair holding the two detections of the same transmission detected by the two receivers. We have calculated
the difference between the two timings. In more detail, lets say the transmission actually happened in time $t_0$. It was detected in receiver A in time $t^A_0$ and in receiver B in time $t^B_0$. Then we calculate the difference in the time of arrival, between receiver A and B:

$$t^{AB}_0 = t^A_0 - t^B_0$$

see figure 9.1 for a plot of $t_i^{AB}$ values of one of the transmission types. Theoretically, since the two receivers’ antennas are at most 10 cm away from each other, the difference should be at maximum at about 0.3 nanoseconds.

The results showed a slowly changing difference between the stations, growing and shrinking with time and going up to hundreds of nanoseconds. This slow drift also carried a much faster changing difference, of a few nanoseconds in size.

Since the clocks in the receivers are “disciplined” by their GPS, we have repeated the experiment with the change that the two receivers shared the same GPS antenna. In the results we found that the slow drift was gone but there was still an average bias towards one of the stations of up to 20 nanoseconds (see figure 9.2). Each transmission had a bit different average bias size, which means it depends also on the signal. We have also conducted experiments in which the receivers have shared the same RF antenna. The results showed there is no effect to this sharing.

These results have showed us that the clocks are not accurate enough for our purposes (we can tolerate only a few nanoseconds of error), but since we can use relative timings (described next) we didn’t investigate further the cause of this inaccuracy.

9.1.2 Clocks Frequency Accuracy and Stability

The ATLAS system has no need in accurate time values, since the localization algorithm does not use the actual time of the detection of a signal, but rather a time interval between the detection and some previous detection (of a beacon transmitter), usually up to a few seconds before. That is why the frequency accuracy and stability of the clocks are the ones which are important for accurate localizations.

In order to test stability we tested whether the two receivers report the same
time interval between each two consecutive transmissions. In more detail, assume two consecutive transmissions that happened in $t_0$ and $t_1$. Each transmission is detected in the two receivers, A and B, and so we have $t^A_0$ - time of transmission 0 as reported in receiver A; $t^B_0$ - time of transmission 0 as reported in receiver B; $t^A_1$ - time of transmission 1 as reported in receiver A and $t^B_1$ - time of transmission 1 as reported in receiver B. We define the time interval of each station:

$$t^A_{0,1} \equiv t^A_1 - t^A_0$$

$$t^B_{0,1} \equiv t^B_1 - t^B_0$$

And the difference of that interval as reported by two stations:

$$t^{AB}_{0,1} \equiv t^A_{0,1} - t^B_{0,1}$$

Which is equal to the difference of two consecutive differences, as defined in
Figure 9.2: The difference of timings of a transmission, made by two receivers, with the stations sharing the same GPS antenna. For this tag there is a constant average bias of about 5 nanoseconds for one of the receivers, with maximum difference of up to about 15 nanoseconds.

The previous subsection:

$$t^{AB}_{0,1} = (t^A_1 - t^A_0) - (t^B_1 - t^B_0)$$

$$= (t^A_1 - t^B_1) - (t^A_0 - t^B_0)$$

$$= t^{AB}_1 - t^{AB}_0$$

The same that the time-of-arrival differences should be zero theoretically, the differences of the time-of-arrival intervals should also be zero. Unlike the time-of-arrival values, the intervals’ differences are due to different clock rates, and not due to wrong clock values.

The statistic calculation we chose in order to compare the clocks stability is the standard deviation of the $t^{AB}_{i,i+1}$ values of the experiment. The maximum value of
these is not a good statistic value to compare since there are rare outliers with large values. The average also does not give useful information because in all of the experiment runs the average was close to zero; This actually shows that the clocks frequencies are averagely the same and they do not drift away. Since theoretically the difference between any two matching intervals should be zero, the smaller the STD of results of a certain transmission, the more they are accurate.

The experiments showed results with interval differences of up to tens of nano-seconds. The standard deviations of the data sets of the different transmissions were between 3 to 27 nanoseconds. An example plot of this calculation is in figure 9.3.

As in the previous experiment, we have also conducted measurements in which the receivers shared the same GPS and in which they shared the same RF antenna. The results can be seen in figure 9.4. It can be seen that sharing the RF antenna yields more accuracy than sharing a GPS. It suggests that the GPS inaccuracy is only in its absolute values and not in the frequency of its clock. The larger accuracy differences that are seen when comparing the shared/unshared RF antenna experiments suggest that the differences in the antennas cause inaccuracies of tens of nano-seconds in the detection times. Another observation is that in general, as expected, longer codes produce more accurate results.

We have also calculated the TOA-interval differences in the real receiver stations already working in the Hula Valley. In this case, the receivers are in distances of few kilometers from each other. Still, the differences of intervals of times-of-arrival between the receivers should be zero, as in the lab, since the signal propagation velocity is constant.

We have, for now, 5 stations which make 10 unique station pairs. We have queried the database for 3 hours of detections of a specific transmission which has a period of two seconds. We have calculated the difference of intervals for every pair. figure 9.3 shows the average values of these interval-differences, compared to the distance between the stations to the tag. It can be seen that the closer the stations are to the tag, the more they are accurate. It seems reasonable that this is because of the signal strength attenuation over distance. It should be noticed that unlike the lab experiments described in the previous part of this section, the field receivers have additions of custom antennas and filters, as detailed in section 4.1.
Figure 9.3: Difference of each interval of time between consecutive transmissions, between two stations. The standard deviation of the difference is 10.07 nanoseconds.

9.2 Localization Field Results

9.2.1 Standard Deviation of Localization of a Stationary Tag

As a measurement of the system’s localization stability and consistency we examine a period of time of a specific stationary tag’s localizations. Ideally the variance should be zero as all the localizations should have the same value. In reality, of course, the values are spread around the real location in a shape of an ellipse. The standard deviation (STD) of the localizations is dependent on the direction. We find the azimuth with maximum STD in order to represent the consistency of the system. The STD values are calculated using the X-Y plane only, without considering altitude, since it has less significance in the variance. Figure 9.6 shows maximum STD of two tags in the Hula Valley, calculated over 10 hours of the system work data (A transmission once every two seconds), when detection was made by all of the five receiver stations in the field. The STD values are around 10 meters. It should be said that when including localizations made with detections from less receiver stations, the STD values grow, meaning, less consistency in the
Figure 9.4: Average difference of intervals of times-of-arrival in two receivers with different setups

system.

9.2.2 Visualization of Wildlife Tracking

The system has been tested on wildlife in the Hula Valley. Detailed description of these research projects will appear in future publications. We include here only a few visualizations of the localizations made by the system. Figure 9.7 shows screen shots of Google-Maps showing localization data that was taken from the system’s database.

9.3 Simulation

ATLAS includes a simulation program, described in 6.3. One of its purposes is testing the localization algorithm. It saves the locations of the made-up path to a file. The Matlab localization program can be executed in a special simulation mode in which for every localization it estimates, it also reads from the file the real location that generated the localization task. It can then visualize graphically the real trail and the estimated trail, as shown in figures 9.8 and 9.9.
Figure 9.5: Average difference of detection intervals between 10 pairs of stations, compared to the average distance of each station pair to the tag.

Figure 9.6: Standard deviations of localizations of tags over 10 hours: Localization data of 10 hours of two tags, done by 5 receiver stations. The line shows the azimuth with the maximum standard deviation of the localization solutions.
Figure 9.7: Visualization of wildlife tracking: Localizations of tags put on a Kestrel and a Barn Owl in the Hula Valley. The different colors represent adjacent detection times. Each image shows up to 2000 detections, which come from a period of up to 48 hours of the system’s work.
Figure 9.8: Simulator of three stations: two simulated trails and their localization results, with 3 stations. Erroneous results are the where the estimation marker (blue circle) is not on top of the real location marker (green x). It can be observed that the locations where the algorithm fails are related to the geometry of the base stations locations.

Figure 9.9: Simulator of four stations: a simulated trail and its localization results, with 4 stations. This simulation had no major erroneous estimations - all estimation markers are on top of the real location markers.
References


תקציר

זה תזה המתארת את עיצובה וימושה של מערכת ATLAS, מערכת לאיכון naval ב Brigham ג'ופר בישופ והחלמתי שהיתרונות העיקריים שבימוש במערכת GPS ביחס לушки_nf או LES יש עולם בו הוא מגלגל מדמצת המאסטר הדרה ומערכת השידור המ햤ית וה/animationsים מאוחרות גם במובארה, בעלות התנה רוב המאדורגנברג מתוארים ומתחמות והPEndPointים באתה והשידורים שמבחנה את העולם והחיים של המים, וראיתו ונייזר וגו בבל המשפטים Bluetoothים portfolios מצומצם במע方もים גם הזדמנויות cut recovers המձומרים אל-לאירוט (לערכט מיקום), או אלגוריתמי התת-חומרות הביני החותרון ב- ATLAS מתחמות חוזרים. המורכבות הנפשו התיהו בנב אלטרת החולות אושר בצפריע של ישראל, רכז עס_PATH תחתון קולות, היא נמסה עד יתי לרידת השחזרות וד לאכמה גים שובים של בול חים. ו MOZAT되었습니다 מסופק החירה ישר בתוקף, קבוצת מחסנית של מודיע החיה.
מערבת אייקון ATLAS ליעול הסרתי

יעזוב ומימוח

תזה המוגשת ל FETCHillian תדרישות קבלת התואר
"מוסמך למידעים" במדעי המחשב

מאת
אורן קישון

עבדת המחקר בועزة בהנחיית השל
פרופ' סיון טולדו

ינואר 2015
שבט’ התשע”ה