Theodolite notice of use

Maxime Vaidis

June 10, 2020

Version	*Date*	*Comments*
1.0	June 10, 2020	Initial writing

1 Introduction

In mobile robotics, it may be important to have reference data in order to compare and validate the results obtained by an algorithm. This comparison is important for assessing the location of a mobile platform. Indeed, sensors such as Global Position System (GPS) have an inaccuracy which can be several meters and cannot be a reliable source to accurately evaluate a location algorithm. Algorithms such as Iterative Closest Point (ICP) are more accurate, but have long-term biases which distort location.

To overcome this problem of inaccuracy measurements, some trades such topographers or surveyors use devices called theodolite. A theodolite is a geodetic instrument supplemented by an optical instrument, which makes it possible to measure angles in both horizontal and vertical planes in order to determine a direction. These devices can also measure the distance of a target from them. Their angular accuracy is less than one thousandth of a degree, and the accuracy of the distance measurement is of the order of a millimeter.

As a result, it is possible to obtain a very accurate survey of any target objects. By placing different targets on a mobile robot, we could then get its full reference pose. These baseline data can then be used to assess ICP accuracy and quantify its bias over time.

2 Equipements

In order to carry out our surveys, we will need several equipment in addition to the theodolite. These are means of communication between the theodolite and the data acquisition platform, but also the different targets used to accurately measure distance. These different equipment are presented in this section.

2.1 Theodolite

In this section, a precise description of the theodolite will be given, as well as the explanation of its start-up for the measurement. The theodolite we use is a Trimble Total Station S7, shown in Figure 1.

The data sheet for this theodolite model is given in this link Technical Guide French or in this link Technical Guide English. A short description of the abilities of the theodolite is given. The accuracy and performance of the different sensors are given, as well as the duration of the batteries.

A quick start guide for this template is also given in this link Quick Start Guide French or in this link Quick Start Guide English. In this guide, a description of the equipment of the theodolite is given. The first hand of the equipment is also explained in order to start the theodolite for the measurement.

With this theodolite, other useful accessories can be used. The laboratory has a digital tablet that can be attached to the theodolite, shown in Figure 2.

This tablet has a windows OS with Trimble Access software that allows communicating commands to the theodolite. A detailed description of this software will be given in the Section 3. The theodolite can take into account different means of communication: Bluetooth, Radio and USB. Bluetooth mode can be direct (computer to theodolite via Trimble Access). This Bluetooth mode can also be used for indirect communication

Figure 1: Trimble S7 of the laboratory.

Figure 2: Tablet fixed on the trimble S7.

to the theodolite. In this case, a Trimble TDL 2.4 relay should be used to link the computer to the theodolite. The computer must connect to this relay via Bluetooth, and via the Trimble Access application. Afterwards, the relay will communicate by radio with the theodolite, which allows to control it remotely (up to about 600 meters). The last mode of communication is via USB. This is the one we will use mainly afterwards. All these means of communication will be described in detail with the presentation of the Trimble Access software in the Section 3.

2.2 Target

In order to perform measurements, the theodolite needs a target. These targets can be of different types. The most common is the reflective foil target. The Figure 3 shows some examples.

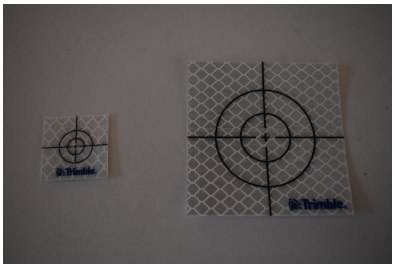

Figure 3: Tablets which can be used with the Trimble.

These targets are provided with the theodolite and can be placed on any object. However the theodolite can only measure one target at a time. Theodolite makes no distinction between these different reflective targets. Thus, if it detects a new target near another target, it can change the target to be measured automatically. Moreover, according to the data sheet of the previous section, the performance is lower with this type of target (reduced distance and precision). This is why we prefer to use them to calibrate theodolite over short distances, rather than to use them for distant measurements.

The target that we will most often used to track our robotic platform is a prism, shown to Figure 4.

Figure 4: Prism used as target with trimble S7.

This prism has many assets. First, it is possible to ask a theodolite to lock onto it. Theodolite will not change target during measurement. The prism contains a unique digital signature that guarantees this lock. This makes it possible to place several prisms on the same robotic platform while guaranteeing the correct measurement. In addition, the performance of the measurements is increased. It is possible to follow the prism over longer distances with increased precision. Finally, it can be fixed securely to a platform via a screw. The prism works with a battery that activates its electronic signature. It is possible to set the number of this signature (ranging from 1 to 8) to differentiate it from other prisms. This target is quite heavy and expensive. When used, it should be taken care of and used safely to avoid damage. A paper leaflet is provided with the equipment, illustrating its use and the precautions to be taken.

2.3 Communication part

Communication of data to a robotic platform becomes difficult with the equipment supplied with the theodolite. The USB or Bluetooth is not suitable, while the relay is only usable under the Trimble Access application that works on Windows exclusively. We decided to use another communication module to solve this problem. It is a Lora antenna communication module on a printed circuit board that can be configured to transmit data over long ranges. This module will be installed on a Raspeberry Pi 3B+ board, which will receive the theodolite data via USB. The robotic platform will also be equipped with a Raspberry Pi 3B+ and a Lora module to communicate with the theodolite, which will allow us to save the data at distance.

The LoRa module is presented in Figure 5.

This website provides an overview of the possibilities offered by this module, as well as some code examples for some applications.

The following document Lora GPS HAT Manual provides more detailed examples of use and a start guide for use. Additional information will be given in the Section 4 and Section 5.

Figure 5: LoRa antenna and it shield used on a Rspberry Pi 3B+.

2.4 Raspberry Pi 3B+

The computer currently used to interface with the theodolite is a Raspberry Pi 3 model B+ running Ubuntu Mate 18.04.2. Using another computer and operating system should theoretically work, as long as the OS is a Linux distribution and as long as the computer has a 32-bit ARM based processor. The only computers that have been tested are Raspberry Pi 3 model B/B+ and the only OS's that have been tested are Ubuntu Mate 18.04.2 and Raspbian 9.11 Stretch.

The credentials for logging in the computer are the following:

- User: pi
- Password: macaroni

3 Software

The configuration of the theodolite can be done in two different ways. The easiest and fastest to implement is using the Trimble Access application. This application is only available on Windows, but allows using all the functionalities of the theodolite quite quickly. The second way is to use C++ code libraries that allow communicating with the theodolite. These libraries can be used under Linux, but do not contain all the possible features under Trimble Access. Moreover, it is difficult to implement it in some cases. This section presents these different ways of configuring the theodolite.

3.1 Trimble Access software (coming soon)

Description des possibilit offertes par le logiciel de Trimble Dtailler et montrer les fonctionnalits les plus couramment utilises

3.2 Compile library for linux

The company Cansel has provided us a pre-completed libraries of C++ code to use some of the theodolite features on Linux.

This repository is built upon a project that was intended to be used as a Telnet server for interfacing with a Trimble TotalStation robotic theodolite. Its main program was modified heavily by removing the telnet server

functionality and replacing it with a sequence of simple commands that are sent to the theodolite. This was done in order to test and explore what type of interfacing could be done with the theodolites we have at the lab, which are 3 Trimble TotalStation S7. In other words, the main program now consists of a small sequence of commands that are sent to the theodolite before terminating.

In the theodolite_node_ROS package, these libraries are in the *lib* directory. The *.h* header files that use its libraries are in the *include* directory. To use these precompiled libraries, *.cpp* and *.h* files were created in the *src* directory. These files are from an SDK provided by Trimble. We will detail this in future subsections. The functions defined in these files will be explained, mainly those we used to obtain the data of the theodolite. The two main classes that we will use are Observationlistener and Ssiinstrument, present in the *src* directory.

3.2.1 TPSDK

The main program can access the functionalities of the theodolite through the use of the Trimble Precision SDK (TPSDK). This SDK is provided by the manufacturer of the theodolite in order to allow the theodolite to be integrated into third party software solution.

The SDK is made to be used with Win32 and Windows Mobile platforms. However, this repo uses a special non-official precompiled version of this SDK that is meant to be used on a Raspberry Pi running Raspbian. In our case, the SDK consists of .so library files and .h header files that are located in the lib and include directories, respectively. The source code in this repo is written in C++. The main program, which has dependencies to the SDK, has been compiled and used on a Raspberry Pi 3 model B+ running both Raspbian 9.11 Stretch and Ubuntu Mate 18.04.2.

More detailed explanations and description of the SDK are available on the TPSDK website. To sign in, click Sign In in Trimble Access and enter the following info:

- Username: ULaval
- Organization: trimble-precision-sdk
- Password: ulav123!

On this website you will find a complete description of the SDK, as well as guides and lists of features that are accessible with the SDK. More importantly, you will find the list of the classes in the SDK, along with their methods, attributes and interfaces. It is important to know that the documentation on this website may not correspond exactly to the version used in this repo, since we use a C++ ARM based version of the SDK, instead of the Microsoft .NET version. For this reason, the interfaces might not be the same and they might not be directly usable via C++. Also, since we are not sure about where our version of the SDK is situated in the version release history, some features described in the documentation may not be available in our version of the SDK.

3.2.2 Communication

The Trimble TotalStation SSeries supports many types of communication channels, such as Bluetooth, radio, serial, USB, WiFi, etc. However, we were not able to connect the Raspberry Pi to the TotalStation S7 via Bluetooth or WiFi. The lab has in its possession three Trimble TDL2.4 data link radios, which connect to a computer via Bluetooth and connect to a theodolite via 2.4 GHz radio channel. They could prove useful for long-distance communication between the Raspberry Pi and the theodolite, but we have not been able yet to connect them to a Raspberry Pi and use them. They currently only work with the tablet controller of the theodolite.

This repo currently only supports communication between the Raspberry Pi and the theodolite via a USB and TCP connection. The theodolite shows up in Linux as a USB device with idProduct: 0101, idVendor: 099e and Trimble as manufacturers. A udev rule was created in Ubuntu Mate in order to give users access to the theodolite USB connection. Any user in the group users can run the program and communicate with the device. The TCP connection that was already implemented has not been used or tested.

Since the SDK was made to be used on Windows, the default classes for instantiating a communication channel with the theodolite don't work on Linux. The SDK supports custom communication type, as long as you define a custom communication class which inherits from ICommunicator and implements the methods defined in the ICommunicator interface. The classes USBCommunicator and TCPCommunicator are two implementations of this interface. They are used in the SsiInstrument class in the Connect method.

3.2.3 SsiInstrument

The class SsiInstrument represents the TotalStation theodolite at a high level of abstraction. It implements methods that start specific tasks with a few number of parameters. These methods handle the required sequence of function calls specific to the SDK.

In order to use the features that are implemented in the SsiInstrument class and start interfacing with the theodolite, one must first instantiate a SsiInstrument object and load the appropriate driver. The default loaded driver is a driver for the TotalStation SSeries theodolites. The method to call is SsiInstrument::LoadDriver. It essentially instantiates the appropriate device driver from its DriverManager attribute, which is instantiated in the initialization list in the constructor of the class. The driver is an SSI::IDriver object that is kept as an attribute of the class.

When the driver is loaded, the SsiInstrument::Connect should be called in order to establish a connection to the device. Two parameters can be passed to this function: const char* sType and const char* sPort. The sType parameter specifies the type of communication channel to use. The following types can be specified: "usb", "tcp" and "int". The USB and TCP types use the previously mentioned implementations of the ICommunicator interface: USBCommunicator and TCPCommunicator. USBCommunicator uses the external library libusb-1.0 to connect to the device. The int type of connection stands for internal. It is not clear what this type of connection means and connecting to the device with this type of connection hasn't been tested. The parameter sPort is used to specify the TCP port to be used if the type of connection is TCP. Again, this type of connection has also not been tested. According to the user manual, the TotatStation S7 doesn't seem to support connection via an IP network. The SsiInstrument::Connect method instantiates a SSI::ISensor object and keeps it as an attribute of the class that can be used to access interfaces.

Once the driver is loaded and the device is connected, the user can now call functions to interface with the device. Basic features such as starting video streaming, taking measures, tracking targets and setting the servos angles have been implemented.

In order to access an instrument feature, the SSI::ISensor::getInterface method of the SSI::ISensor object (_pSensor attribute) has to be called while passing it the interface of the feature you want to access. Once you have the interface, you can call the methods defined by the interface to start interacting with the device. Documentation for the interfaces is available on the TPSDK website.

Several tracking modes are defined in the SsiInstrument class. These are: PRISM MODE, AUTOLOCK MODE, DR MODE, DR LASER MODE and MULTITRACK MODE. PRISM MODE allows you to find a prism to make a measurement afterwards. AUTOLOCK MODE ensures that the theodolite locks on to a target as soon as it finds it, and can change target if it finds another one. It is a tracking mode without measurement. DR MODE allows you to lock on a target and perform a measurement only. DR LASER MODE performs the same task as DR MODE, but in addition a powerful laser beam is used to visualize the measurement area. Finally, MULTITRACK MODE allows the theodolite to lock onto a prism and track it while taking measurements at a predefined frequency. It is this mode that we will use to take trajectory references.

To activate a mode, we use the *Target* function. This function takes as input the mode chosen, as well as the number of the target which is in our case the number of the prism that we have selected. Subsequently, tracking is activated by the *Tracking* function which takes as an argument the information started/stopped, and an object of the Observationlistener class to save the positioning data. Once tracking is enabled, the received data is automatically stored in this class. To access it, just use the functions of the Observationlistener class.

3.2.4 ObservationListener

This class is one of the most important for our usage. It defines a vector element which will contain all the data gather. This vector is named *observations*, and his size is defined as $size_vector$. The function *getObservations* and *getSizeVector* will be used several times to have access to the data.

The index of the vector *observations* is defined by an enumeration list. Five topics are listed: HORIZONTAL_ANGLE_VECTOR, VERTICAL_ANGLE_VECTOR, DISTANCE_VECTOR, TIMESTAMP_VECTOR and ERROR. These topics are updated in the function *observationTracked*. This function monitors an event of the theodolite and save the data received. The first two ones are the data of horizontal angle and vertical angle of the theodolite. These data are expressed in arcsecond units, cast into a double for an easier storage. The DISTANCE_VECTOR expresses the distance of the measurements in meter. If the target is not detected or too close to the theodolite, the value will be zero. The timestamps are stored in TIMESTAMP_VECTOR. These data used UTC time as timestamps, so it will be the clock of the computer. The time is expressed as a double. The frequency of the measurement will depend on the configuration choose for the tracking. Finally, ERROR is a flag which will tell the user if there are some issues to collect the data. The different flags are presented is the Table 1.

Table 1: Flag error which can occur during a tracking.

Flag Number	Meaning	Causes	Effect
0	No error occur	Good trackig	Data save correctly
1	TiltOutOfRange	Theodolite not balanced	No data saved
2	WrongTargetDistance	Issue with the target	No data saved
3	Unknown	Unknow issue	No data saved
4	Tracking issues	Target too close or not detected	No data saved

If the user chooses a finite number of measurements, he can use the function *saveFile* to store the data in a CSV file.

3.2.5 Others class functions

The other classes in the *src* directory are not currently used, or allow the functions of the two previous classes to be applied. These are SsiCallbacks, SsiCommand, TCPCommunicator, TCPServer, USBCommunicator and VideoStreamingListener. SsiCallbacks have functions that allow the status of certain configurations. For example, we can ask to know the tilt of the theodolite or if the tracking is still in progress. SsiCommand is a class that sends the commands requested to the theodolite (for example *connect* or *target* among others). TCPCommunicator and TCPServer are classes that allow us to communicate with the theodolite via TCP links. We will not use them in this project. USBCommunicator allows us to communicate by USB with the theodolite. Finally, VideoStreamingListener is a class that has functions that can use the theodolite camera. We will not use it in this project as well.

3.2.6 Remark

The C++ library cannot be used at the same time as the Trimble Access software. Indeed, as soon as one or the other of the software is started, the USB port is busy and cannot connect to other programs to receive information other than those requested by the software used. Therefore, the Trimble Access software should be used in the first instance to calibrate and configure the platform as desired. Subsequently, the program under linux will provide us with the data acquires.

Some features require receiving data asynchronously from the device. Methods like SsiInstrument::Video and SsiInstrument::Tracking require passing listeners that are classes that inherit and implement specific interfaces. For example, in order to access the video streaming frames and the streaming states when calling the SsiInstrument::Video method, one has to pass an object that implements SSI::IVideoStreamingUpdateListener and SSI::IStreamingStateChangedListener. The methods defined by these interfaces are called whenever an event happens. Data is accessible in an event object (such as SSI::VideoStreamingUpdateEvent) that is passed as a parameter in the callbacks defined in the interface.

In our case, two listeners have been developed in order to be able to access tracking measures and video streaming frames. These listeners are ObservationListener and VideoStreamingListener respectively. All they currently do is store the current image or store the received measurements in a vector. It is also possible to save the received measurements to a .csv file. Initially, the repo came with the SsiCallbacks class. This class implemented some listeners that were useful with the Telnet server that was implemented. All of these libraries are compiled under ROS to tune the code we want to use. The next section is about two nodes we have made: one to send the data through LoRa antennas, and one to receive these data again through LoRa antennas.

4 ROS node

ROS (Robot Operating System) is widely used in robotics. It is a meta-library that contains code packages that can be used on different platforms, mainly under Linux. This library allows launching code nodes that can communicate information between them through publisher and subscriber. In order to be able to more easily configure our code to recover the data from the theodolite, we will use a ROS node which will aim to send the data via the Lora antennas, and another ROS node that will receive them and record them on the robotic platform. This section will describe how these two nodes work. The package that contains these two nodes is named theodolite_node_ROS.

4.1 Acquisition of data and sender

In the theodolite_node_ROS package and in the *src* directory, there is the ROS node that sends the data from the theodolite to the receiving Lora antennas. The corresponding file is named *theodolite_node.cpp*. The different parameters of the communication with the Lora antennas as well as the functions used with it are defined just before the main function *main*. This *main* function consists of a while loop that will perform several tasks in a row.

First we get the settings of the configuration we want. These parameters are defined in the launchfile *theodolite_node.launch* present in the *launch* directory of the package. Six parameters can be defined. The number of the theodolite (*theodolite_number*) you wish to use must be entered in a first one. We pick this number. It must be different from the other theodolites used if there are several. This issue will be used to sort the collected data, especially in the case where several theodolite track the same target. The number must be between 1 and 8 inclusively. Then you have to give the number of the prism you want to track, *target_prism*. This number will also be used to sort the data. The third parameter, $number\_of\_measurments$, to be filled in is the number of measurements you want to perform. If you want to make a continuous measurement over time,

you have to set the value to 0. The program will then search for the data each time a new one appears. The user will then have to manually close the node via the terminal. The *use_lora* parameter tells the program if you want to send the data via the Lora antennas. The *show_data* parameter can be used to debugger by displaying the data received from the theodolite on the terminal. Finally, it is possible to save a finite number of data using the *save_measurements* and $file\_measurements$ parameters, which indicate if you want to save the data and the file name, respectively. Note that to save data, the $number\_of\_measurments$ parameters must be different from 0. In addition, only *.csv* is used.

The code then executes the LoRa antenna initialization functions if the option is enabled. A check of the parameter values is then performed to ensure that there are no errors. Theodolite drivers are then loaded with the *LoadDriver* function, and the connection is activated by the *Connect* function. If there is no error at this level, we give the tracking mode to the theodolite via the $Target$ function. Then we define our vector *observation_listener* which will contain the data received from the theodolite. The tracking with the data takes starts with the call of the $Tracking$ function. The code is divided into two parts: one for continuous measurement, and another for a given number of measurements.

If the measurement is continuous, an endless while loop is used to acquire the data. The number of data in our *observation_listener* vector is then checked regularly to detect if new values have been recorded. A 30 millisecond pause is taken between each check to allow time for the program to update. This delay is smaller than the theoretical maximum frequency at which the theodolite can transmit data (20Hz). As a result, the program will not lack new data. When new data are detected, they are accessible via our vector. When *show_data* is enabled, the values are displayed on the terminal. If *use_lora* is enabled, the data is cast into a string variable, and sent byte by byte to the LoRa antennas, which will broadcast it. For a finished measurement, the same procedure is applied, except that the while loop ends when the desired number of measurements is reached. After this number is reached, the program saves the data if the user has enabled the option, then the ROS node logs out of the theodolite with the $Freedriver$ function.

There are seven data that are sent during communication with the robotic platform. The first value is the theodolite number, followed by the prism number in order to differentiate more type of measurement between them. Then the angular measurements are sent: the horizontal angle and the vertical angle. The distance from the prism is the fifth data sent, followed by the timestamp of the angular measurements and distance. Finally, an error flag set to Section 3 is sent last in the character string.

It is possible that connection or other errors occur during the running of the program. The Section 6 is dedicated to this, and provides solutions to solve the problems that have arisen. Now that we can send the data, the code to receive it is described in the next subsection.

4.2 Receiver

Like the previous ROS node, the data listening node is located in the theodolite_node_ROS package and in the *src* directory. The corresponding file is named *theodolite_listener.cpp*. The different parameters of the communication with the Lora antenna as well as the functions used with it are defined just before the main function *main*. This *main* function also consists of a while loop that will perform several tasks as a result.

First we get the settings of the configuration we want. These parameters are defined in the launchfile *theodolite_listener.launch* present in the *launch* directory of the package. Two parameters are used. The first parameter *rate* is used to define the frequency of the reception of messages. The second parameter *show_data* is used to display data received on a terminal to debug or verify that the communication is working properly. Once the parameters have been set, the Lora antenna is configured as for the transmitter code. The difference is that we use an endless while loop, in order to listen at all times to the reception of a message at the frequency defined by the user.

This endless while loop runs the *receivepacket* function, which will read the received bytes and convert them to double format. The data received is then saved in a vector that will be transmitted to ROS for a publisher

named *data_publish*. Afterwards, it is enough to save these data in a rosbag to be able to use them in post-processing. It should be noted that this publisher does not use ROS timestamp. Indeed, it is too complicated to synchronize the data from the theodolite with the ROS clock in real time. This synchronization will have to be done in post-processing. In addition, this code currently only works for a single transmitter-receiver communication. Tests must be carried out to validate it with several transmitters to a receiver.

Again, it is possible that connection or other errors occur during the running of the program. The Section 6 is dedicated to this, and provides solutions to solve the problems that have arisen.

5 Deployment (Coming soon)

Courte introduction pour le dploiement de diffrents theodolite

5.1 Calibration of theodolite

Mise en place sur trpied
Rglage du tilt
Autres ?

5.2 Resection

Pourquoi ? Comment dfinir le repre du theodolite avec une/plusieurs cibles
Si pas de prism ?

5.3 ROS node receiver

Mise en marche du code pour le receiver

5.4 ROS node sender

Mise en marche du code pour le sender

6 Issue/error (Coming soon)

Table des erreurs pouvant survenir durant le dploiement et les solutions apporter