

# Theodolite notice of use

Maxime Vaidis  
June 4, 2020

Version	Date	Comments
1.0	June 4, 2020	Initial writing

## 1 Introduction

En robotique mobile, il peut tre important d'avoir des donnees de rfrence afin de comparer et valider les rsultats obtenus par un algorithme. Cette comparaison est d'autant plus importante pour valuer la localisation d'une plateforme mobile. En effet, les capteurs tels que les GPS ont une imprcision pouvant tre de plusieurs mtres et ne peuvent pas tre une source fiable pour valuer finement un algorithme de localisation. Les algorithmes tels que ICP sont plus prcis, mais possdent des biais sur le long termes qui faussent la localisation.

Pour palier ce problme d'imprcision des mesures, certains corps de mtiers comme les topographes ou les gomtre ont recours des appareils appels thodolite. Un thodolite est un instrument de gdsie complet dun instrument d'optique, qui permet de mesurer des angles dans les deux plans horizontaux et verticaux afin de dterminer une direction. Ces appareils peuvent galement mesurer la distance d'une cible par rapport eux. Leur prcision angulaire est infrieur au millime de degr, et la prcision de la mesure de distance est de l'ordre du millimtre.

De ce fait, il est possible d'obtenir un relev trs prcis de n'importe quels objets cibls. En plaant diffrentes cibles sur un robot mobile, nous pourrions alors obtenir sa pose complete de rfrence. Ces donnees de rfrence peuvent alors nous servir valuer la prcision d'ICP et quantifier son biais dans le temps.

## 2 Equipements

Afin d'effectuer nos relevs, nous aurons besoin de plusieurs quipements en plus du thodolite. Il s'agit de moyens de communication entre le thodolite et la plateforme d'acquisition des donnees, mais galement des diffrents cibles utiliss pour mesurer la distance avec prcision. Ces diffrents quipements sont prsents dans cette section.

### 2.1 Theodolite

Dans cette section, une description prcise du thodolite sera donn, ainsi que l'explication de sa mise en marche pour la prise de mesure. Le thodolite que nous utilisons est une Station Totale Trimble S7, montr la [Figure 1](#).

La fiche technique de ce modle de thodolite est donn dans ce [Technical Guide French](#) ou dans ce [Technical Guide English](#). Une courte description des capacits du thodolite y est donn. La prcision et la performance des diffrents capteurs y sont renseignes, ainsi que la dure des batteries.

Un guide de dmarrage rapide de ce modle est galement donn dans ce [Quick Start Guide French](#) ou dans ce [Quick Start Guide English](#). Dans ce guide, une description des quipements du thodolite y est donn. La premire prise en main des quipements y est galement expliqu afin de mettre en marche le thodolite pour la prise de mesure.

Avec ce theodolite, d'autres accessoires utiles peuvent tre utiliss. Le laboratoire possde une tablette numrique pouvant tre fix sur le thodolite, montr [Figure 2](#).

Cette tablette possde un OS windows avec le logiciel Trimble Access qui permet de communiquer des commandes au thodolite. Une description dtaille de ce logiciel sera donne dans la [Section 3](#). Le thodolite peut prendre en compte diffrent moyens de communication: Bluetooth, Radio et USB. Le mode Bluetooth peut tre



**Figure 1:** Trimble S7 of the laboratory.



**Figure 2:** Tablet fixed on the trimble S7.

direct (ordinateur vers thodolite via l'application Trimble Access). Ce mode Bluetooth peut aussi tre utilis pour une communication indirect vers le thodolite. Dans ce cas, il faut utiliser un relai Trimble TDL 2.4 qui assurera la liaison entre l'ordinateur et le thodolite. L'ordinateur doit se connecter ce relai par Bluetooth via l'application Trimble Access. Par la suite, le relai va communiquer par radio avec le thodolite, ce qui permet de pouvoir le contrler distance (jusqu' environ 600m). Le dernier mode de communication est par USB. C'est celui que l'on va utiliser principalement par la suite. Tout ces moyens de communication seront dcrit en dtail avec la prsentation du logiciel Trimble Access dans la [Section 3](#).

## 2.2 Target

Afin d'effectuer des mesures, le thodolite a besoin d'une cible. Ces cibles peuvent tre de diffrent types. La plus commune est la cible sur papier rflchissant (reflective foil). La figure ... en montre quelques exemples.

Ces cibles sont fournies avec le thodolite et peuvent tre plac sur n'importe quels objets. Cependant le thodolite ne peut mesurer qu'une cible la fois. Le thodolite ne fait pas de distinction entre ces diffrentes cibles rflchissantes. Ainsi, s'il en dtecte une nouvelle proximit d'une autre cible, il peut changer de cible mesurer automatiquement. De plus, d'apr la fiche technique de la section prcdente les performances sont moindres avec ce type de cible (distance et prcision rduites). C'est pourquoi on prfrera plutt les utiliser pour calibrer le thodolite sur des distances courtes, plutt que de les utiliser pour des prises de mesures lointaines.

La cible que l'on va utilis le plus souvent pour traquer notre plateforme robotique est un prisme, montr la [Figure 3](#).

Ce prisme a de nombreux atout. Premirement, il est possible de demander un thodolite de se verrouiller sur celui-ci. Le thodolite ne changera pas de cible durant la prise de mesure. Le prisme contient une signature numrique unique qui garantie ce verrouillage. Cela permet de placer plusieurs prismes sur une mme plateforme



**Figure 3:** Prism used as target with trimble S7.

robotique tout en garantissant la bonne prise des mesures. De plus, les performances des mesures sont augmentées. Il est possible de suivre le prisme sur de plus longues distances avec une précision accrue. Enfin, il est possible de le fixer solidement une plateforme via une vis. Le prisme fonctionne avec une batterie qui permet d'activer sa signature électronique. Il est possible de régler le numéro de cette signature (allant de 1 à 8) afin de la différencier d'autres prismes. Cette cible est assez lourde et onreuse. Lors de son utilisation, il convient donc d'en prendre soin et de l'utiliser de manière sécuritaire pour éviter de l'endommager. Une notice papier est fournie avec l'équipement, illustrant son utilisation et les précautions à prendre.

### 2.3 Communication part

La communication des données vers une plateforme robotique devient difficile avec les équipements fournis avec le théodolite. L'USB ou le Bluetooth ne sont pas adaptés, tandis que le relais n'est utilisable que sous l'application Trimble Access qui fonctionne sur Windows exclusivement. Nous avons décidé d'utiliser un autre module de communication pour résoudre ce problème. Il s'agit d'un module de communication par antenne LoRa sur un circuit imprimé que l'on peut configurer pour transmettre des données sur de longues portées. Ce module sera installé sur une carte Raspberry Pi 3B+, qui recevra les données du théodolite par USB. La plateforme robotique sera équipée également d'une Raspberry Pi 3B+ et d'un module LoRa pour communiquer avec le théodolite, ce qui nous permettra de sauvegarder les données à distance.

Le module LoRa est présent [Figure 4](#).



**Figure 4:** LoRa antenna and its shield used on a Raspberry Pi 3B+.

Ce site internet donne une vue d'ensemble des possibilits qu'offre ce module, ainsi que quelques exemples de codes pour certaines applications.

Le document suivant [LoRa GPS HAT Manual](#) donne des exemples plus dtail d'utilisation, ainsi qu'un guide de dmarrage pour celles-ci. Des informations complmentaires seront donnes dans la [Section 4](#) et [Section 5](#).

## 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

La configuration du thodolite peut se faire de deux manires diffrentes. La plus pratique et la plus rapide mettre en uvre consiste utiliser l'application Trimble Access. Cette application est uniquement disponible sur Windows, mais permet d'utiliser toutes les fonctionnalits du thodolite assez rapidement. Le second moyen est d'utiliser des librairies C++ de code qui permettent de communiquer avec le thodolite. Ces librairies peuvent tre utilises sous Linux, mais ne contiennent pas toutes les fonctionnalits possibles sous Trimble Access. De plus il est difficile de mettre en uvre celle-ci dans certains cas. Cette section prsente ces diffrents moyens de configurer le thodolite.

### 3.1 Trimble Access software (coming soon)

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

### 3.2 Compile library for linux

L'entreprise Cansel nous a fournit des librairies pr-compiles de code C++ permettant d'utiliser certaines fonctionnalits du thodolite sous 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.

Dans le package theodolite\_node\_ROS, ces librairies se trouvent dans le rpertoire *lib*. Les fichiers d'enttes *.h* qui utilisent ses librairies sont dans le rpertoire *include*. Pour utiliser ces librairies pr-compiles, des fichiers *.cpp* et *.h* ont t crs dans le rpertoire *src*. Ces fichiers sont issues d'un SDK fournit par Trimble. Nous dtaillerons celui-ci dans les sous-sections venir. Les fonctions dfinit dans ces fichiers seront explicits, principalement celles

que nous utilisons pour obtenir les données du théodolite. Les deux principales classes que nous allons utiliser sont ObservationListener et SsiInstrument, présent dans le répertoire *src*.

### 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 3rd 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 to 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 type 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 manufacturer. 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 implementation 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 an 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 initialisation 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 mentionned 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 intrument 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.

Plusieurs mode de tracking sont dfinis dans la classe SsiInstrument. Il s'agit des modes: MODE\_PRISM, MODE\_AUTOLOCK, MODE\_DR, MODE\_DR\_LASER et MODE\_MULTITRACK. MODE\_PRISM permet de trouver un prisme pour effectuer une mesure par la suite. MODE\_AUTOLOCK fait en sorte que le thodolite se verrouille sur une cible ds qu'il la trouve, et peut changer de cible s'il en trouve une autre. C'est un mode de suivi sans prise de mesure. MODE\_DR permet de ce verrouiller sur une cible et d'effectuer une mesure uniquement. MODE\_DR\_LASER effectue la mme tche que MODE\_DR, mais en plus un puissant rayon laser est utilis afin de visualiser la zone de prise de mesure. Enfin, MODE\_MULTITRACK permet au thodolite de se verrouiller sur un prisme et d'effectuer un suivi de celui-ci tout en prenant des mesures une frquence pr-dfini. C'est ce mode que nous allons utiliser pour prendre des rfrences de trajectoires.

Pour activer un mode, on utilise la fonction *Target*. Cette fonction prend en entre le mode choisi, ainsi que le numro de la cible qui est dans notre cas le numro du prisme que l'on a slectionn. Par la suite, le tracking est activ par la fonction *Tracking* qui prend en argument l'information dmarr/arrêt, et un objet de la classe ObservationListener pour sauvegarder les donnes de positionnement. Une fois le tracking activ, les donnes reues sont automatiquement stockes dans cette classe. Pour y avoir accs, il suffit d'utiliser les fonctions de la classe ObservationListener.

### 3.2.4 ObservationListener

This class is one of the most important for our usage. It define a vector element which will contain all the data gather. This vector is named *observations*, and his size is defined as *sizeVector*. 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 two first one are the data of horizontale angle and verticale angle of the trimble. These data are expressed in arc-second unit, cast into a double for an easier storage. The DISTANCE\_VECTOR expresses the distance of the measurments in meter. If the target is not detected or too close to the trimble, 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 expresses as a double. The frequency of the measurement will depend of 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 ...

If the user choose 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

Les autres classes prsent dans le rpertoire *src* ne sont pas utilises actuellement, ou permettent de faire appliquer les fonctions des deux classes prcdentes. Il s'agit de SsiCallbacks, SsiCommand, TCPCommunicator, TCPServer, USBCommunicator et VideoStreamingListener. SsiCallbacks possdent des fonctions qui permettent l'tat de certaines configurations. Par exemple l'on peut demander connatre le tilt du thodolite ou si le tracking est toujours en cours. SsiCommand est une classe qui permet d'envoyer les commandes demandes au thodolite (par exemple *connect* ou *target* entre-autres). TCPCommunicator et TCPServer sont des classes qui nous autorisent communiquer avec le thodolite par liaison TCP. Nous ne les utiliserons pas dans ce projet. USBCommunicator nous permet de communiquer par USB avec le thodolite. Enfin, VideoStreamingListener est une classe qui possdent des fonctions pouvant utiliser la camra du thodolite. Nous ne l'utiliserons pas galement dans ce projet.

### 3.2.6 Remark

L'utilisation de la librairie C++ ne peut pas se faire en mme temps que l'utilisation du logiciel Trimble Access. EN effet, ds que l'un ou l'autre des logiciels est dmarr, le port USB est occup et ne peut pas se connecter d'autres programme pour recevoir des informations autres que celles demandes par le logiciel utilis. De ce fait, il convient d'utiliser dans un premier le logiciel Trimble Access pour calibrer et configurer la plateforme comme on le souhaite. Par la suite, le programme sous linux nous fournira les donnes acquises.

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 parameter in the callbacks defined in the interface.

In our case, two listeners have been developped in order to be able to access tracking measures and video streaming frames. These listener 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 library are compiled under ROS to parameter the code we want to use. The next section is about two nodes we have made: one to send the data through LoRa antenna, and one to receive these data again through LoRa antenna.

## 4 ROS node

ROS (Robot Operating System) est très utilis en robotique. C'est une grande bibliothèque qui contient des packages de code pouvant être utilis sur différentes plateformes, principalement sous Linux. Cette bibliothèque permet de lancer des nœuds de code pouvant communiquer des informations entre eux travers des publisher et subscriber. Afin de pouvoir paramétrer plus facilement notre code pour récupérer les données du théodolite, nous utiliserons un nœud de ROS qui aura pour but d'envoyer les données via les antennes LoRa, et un autre nœud de ROS qui les recevra pour les enregistrer sur la plateforme robotique. Cette section décrira le fonctionnement de ces deux nœuds. Le package qui contient ces deux nœuds est nommé theodolite\_node.ROS.

### 4.1 Acquisition of data and sender

Dans le package theodolite\_node.ROS et dans le répertoire *src*, se situe le nœud de ROS qui permet d'envoyer les données du théodolite vers l'antenne LoRa réceptrice. Le fichier correspondant est nommé *theodolite\_node.cpp*. Les différents paramètres de la communication avec l'antenne LoRa ainsi que les fonctions utilisées avec celle-ci sont définies juste avant la principale fonction *main*. Cette fonction *main* se compose d'une boucle *while* qui effectuera plusieurs tâches la suite.

Tout d'abord on récupère les paramètres de la configuration que l'on souhaite. Ces paramètres sont définis dans le launchfile *theodolite\_node.launch* présent dans le répertoire *launch* du package. Six paramètres peuvent être définis. Il faut dans une première renseigner le numéro du théodolite (*theodolite\_number*) que l'on souhaite utiliser. Ce numéro, c'est nous qui le choisissons. Il doit être différent des autres théodolites utilisés s'il y en a plusieurs. Ce numéro servira à trier les données recueillies, notamment dans le cas où plusieurs théodolite suivent la même cible. Le numéro doit être compris entre 1 et 8 inclus. Ensuite il faut donner le numéro du prisme que l'on souhaite traquer, *target\_prism*. Ce numéro servira également lors du tri des données. Le troisième paramètre, *number\_of\_measurements*, renseigné est le nombre de mesures que l'on souhaite effectuer. Si l'on veut faire une prise de mesure continue dans le temps, on doit mettre la valeur de 0. Le programme ira alors chercher les données chaque fois qu'une nouvelle apparait. L'utilisateur devra alors fermer le nœud manuellement via le terminal. Le paramètre *use\_lora* indique au programme si l'on veut envoyer les données via les antennes LoRa. Le paramètre *show\_data* peut être utilisé pour debugger en affichant les données reçues du théodolite sur le terminal. Enfin, il est possible de sauvegarder un nombre de données grâce aux paramètres *save\_measurements* et *file\_measurements*, qui indiquent respectivement si l'on veut sauvegarder les données ainsi que le nom du fichier si c'est le cas. Il est à noter que pour sauvegarder des données, le paramètre *number\_of\_measurements* doit être différent de 0. De plus, seul le format *.csv* est utilisé.

Le code exécute par la suite les fonctions d'initialisation de l'antenne LoRa si l'option est active. Un contrôle des valeurs des paramètres est ensuite effectué pour s'assurer qu'il n'y a pas d'erreurs. Les drivers du théodolite sont ensuite chargés avec la fonction *LoadDriver*, et la connexion est active par la fonction *Connect*. S'il n'y a pas d'erreur à ce niveau, on donne le mode de tracking au théodolite via la fonction *Target*. Ensuite on définit notre vecteur *observation\_listener* qui contiendra les données reçues du théodolite. Le tracking avec la prise de données débute avec l'appel de la fonction *Tracking*. Le code se divise en deux parties: une partie pour la prise de mesure continue, et une autre qui effectue un nombre donné de mesures.

Si la prise de mesure est continue, on utilise une boucle *while* sans fin afin d'acquérir les données. Le nombre de données dans notre vecteur *observation\_listener* est alors contrôlé régulièrement afin de détecter si de nouvelles valeurs ont été enregistrées. Une pause de 30 millisecondes est effectuée entre chaque contrôle afin de laisser le temps au

programme de se mettre jour. Ce délai est plus petit que la fréquence maximale théorique laquelle le théodolite peut transmettre des données (20Hz). De ce fait, le programme ne manquera pas de nouvelles données. Lors de la détection de nouvelles données, celles-ci sont accessibles via notre vecteur. Lorsque l'option *show\_data* est active, les valeurs sont affichées sur le terminal. Si *use\_lora* est activé, les données sont mises dans une variable de type string, puis envoyées par byte à l'antenne LoRa qui les diffusera. Pour une prise de mesure finie, la même procédure est appliquée, si ce n'est que la boucle while se termine quand le nombre de mesures voulue est atteint. Après que ce nombre soit atteint, le programme sauvegarde les données si l'utilisateur a activé l'option, puis le nœud de ROS se déconnecte du théodolite avec la fonction *FreeDriver*.

Il y a 7 données qui sont envoyées lors de la communication avec la plateforme robotique. La première valeur est le numéro du théodolite, suivi par le numéro du prisme afin de différencier plus facilement entre elles. Ensuite les mesures angulaires sont envoyées : l'angle horizontale et l'angle verticale. La distance par rapport au prisme est la 5ème donnée envoyée, suivie du timestamp des mesures angulaires et de la distance. Enfin, un flag des erreurs défini la [Section 3](#) est envoyé en dernier dans la chaîne de caractères.

Il est possible que des erreurs de connexion ou autres se produisent durant le fonctionnement du programme. La [Section 6](#) y est consacrée, et apporte des solutions pour résoudre les problèmes survenus. Maintenant que nous pouvons envoyer les données, le code pour les recevoir est décrit dans la sous-section suivante.

## 4.2 Receiver

Comme le précédent nœud de ROS, le nœud d'écoute des données se situe dans le package `theodolite_node_ROS` et dans le répertoire `src`. Le fichier correspondant est nommé `theodolite_listener.cpp`. Les différents paramètres de la communication avec l'antenne LoRa ainsi que les fonctions utilisées avec celle-ci sont définies juste avant la principale fonction `main`. Cette fonction `main` se compose également d'une boucle `while` qui effectuera plusieurs étapes à la suite.

Tout d'abord on récupère les paramètres de la configuration que l'on souhaite. Ces paramètres sont définis dans le launchfile `theodolite_listener.launch` présent dans le répertoire `launch` du package. Deux paramètres y sont utilisés. Le premier paramètre, `rate`, sert à définir la fréquence de l'écoute de la réception des messages. Le second paramètre `show_data` sert à afficher les données reçues sur un terminal afin de debugger ou vérifier que la communication fonctionne correctement. Une fois que les paramètres ont été définis, l'antenne LoRa est configurée comme pour le code de l'émetteur. La différence est que l'on utilise une boucle `while` sans fin, dans le but d'écouter en tout temps la réception d'un message à la fréquence définie par l'utilisateur.

Cette boucle `while` sans fin exécute la fonction `receivepacket`, fonction qui va lire les bytes reçus et les convertir au format double. Les données reçues sont ensuite sauvegardées dans un vecteur qui sera transmis sur ROS pour un publisher nommé `data_publish`. Par la suite, il suffit de sauvegarder ces données dans un rosbag pour pouvoir les utiliser en post-traitement. Il est à noter que ce publisher n'utilise pas de timestamp de ROS. En effet, il est trop compliqué de synchroniser les données issues du théodolite avec l'horloge de ROS en temps réel. Cette synchronisation devra se faire en post-traitement. De plus, ce code ne fonctionne pour le moment que pour une seule communication émetteur-récepteur. Des tests devront être effectués pour le valider avec plusieurs émetteurs vers un récepteur.

## 5 Deployment (Coming soon)

Courte introduction pour le déploiement de différents théodolites

### **5.1 Calibration of theodolite**

Mise en place sur tr pied

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