Terabee Sensors for Robot Alignment

Niall Mullane

September 2019

ZEBRACORNS

FRCTEAM900

A Zebracorn Labs Paper

Contents

1 Introduction

During the 2019 season, we experimented with some exciting sensors from Terabee. Terabee makes a variety of compact Time-of-Flight distance sensors, depth cameras, and thermal cameras. The main sensors that we started using this year were the TeraRanger Multiflex and the TeraRanger Evo 64px. We have been experimenting this season with using these sensors for creating autonomous alignment algorithms.

2 Sensors

2.1 TeraRanger Multiflex

The TeraRanger Multiflex is essentially an array of eight incredibly small and lightweight distance sensors connected by ribbon cable of various lengths. They are really useful for getting synchronized high frequency distance readings from several sources. The flexibility of having up to eight different distance sensors connected by flexible cables allows the Multiflex sensor to be used on basically any platform. It is important to note that the Multiflex cables are quite delicate, so when mounting the sensors, the cables need to be thoughtfully routed.

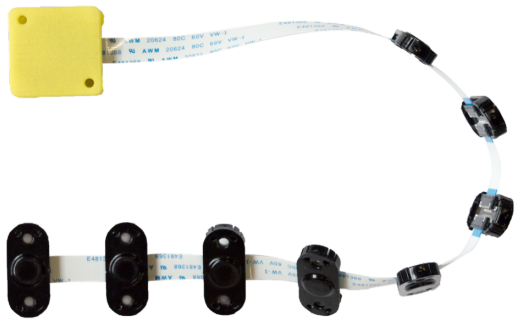

Figure 2.1: Teraranger Multiflex [1]

2.2 TeraRanger Evo 64px

The TeraRanger Evo 64px is a sensor that outputs a 8x8 depth image with a field of view of $15°$and a maximum range of 5m. The Evo is a good way to get coarse high frequency depth data in a single direction. It can operate in two different modes. A fast mode which has an increased data frequency (up to 130hz), but can't get depth readings less than 0.5m, and a close mode which has a decreased data frequency (up to 80hz), but can get depth readings as close as 0.1m.

Figure 2.2: Teraranger Evo 64px [2]

3 Setting up Terabee Sensors

Getting these sensors up and running was an incredibly easy process. The sensors had a premade ROS package that would read data from the sensors and publish it as ROS messages. The ROS package for the TeraRanger Multiflex can be viewed here, and the ROS package for the TeraRanger Evo 64px can be viewed here. To launch the ROS node for the TeraRanger Multiflex, you have to specify the port and which distance sensors you want to read data from. For the TeraRanger Evo 64px, you just launch the ROS

[1] Image from Terabee
[2] Image from Terabee

node with with a portname. Once the nodes are launched all of the distance information will be constantly published on a ROS topic. If you aren't using ROS, there are some simple example programs included in the user manual for the sensor that show you how to interface with the sensor using Python.

4 Alignment Algorithms

Throughout the 2019 FRC season, we experimented with a few different alignment algorithms using these Terabee sensors. The three main algorithms that we developed were aligning to face a flat wall using the Terabee Multiflex, aligning to face the cargo port on the FRC Game Object "The Rocket", and aligning to face the hatch panel port on the FRC Game Objects "The Rocket", "The Cargo Ship", and "The Loading Station".

4.1 Aligning with a Wall

The first algorithm that we experimented with was a simple method for aligning to directly face a wall. We used the TeraRanger Multiflex with two of the distance sensors facing the wall. With these two distance sensors, we ran a simple PID loop on the difference between the two distance values that controlled the angular velocity of the robot. This simple approach allows the robot to continuously track the "angle" of the wall and directly face it. A video of this algorithm can be seen at the following link. https://youtu.be/zl1x0Vbacyw We never used this algorithm on our competition robot during the 2019 season, but it is a pretty effective method for making sure the robot is directly facing a wall or game object.

4.2 Aligning with the FRC Cargo Port on The Rocket

The second algorithm that we developed was for aligning the robot in front of the FRC Game Object called The Rocket. Similar to before, we had two of the TeraRanger Multiflex distance sensors on the corners of the robot. Because of the shape of the rocket the two distance sensors will only give the same distance output when the robot is centered on the cargo face of the rocket as shown in the following image.

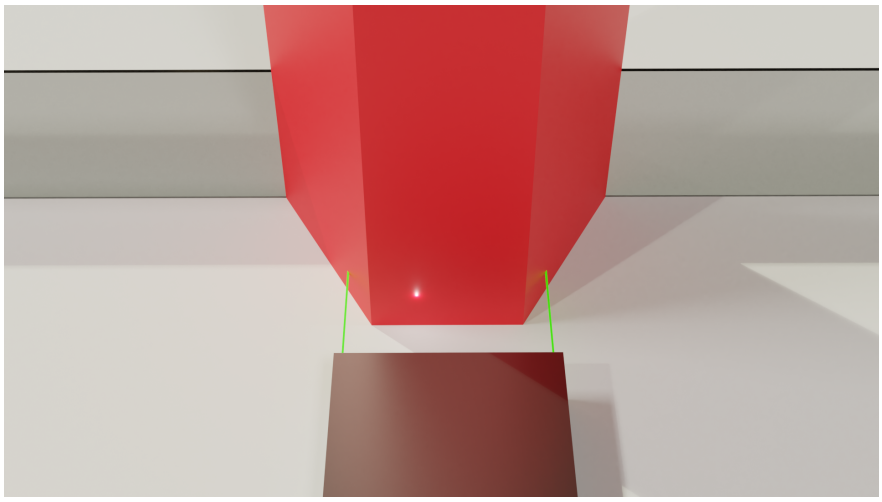

Figure 4.1: A robot in front of the rocket with the beams of the two distance sensors shown

If the robot is off to one side, the distance sensor on that side will output a larger reading than the sensor on the other side as shown below.

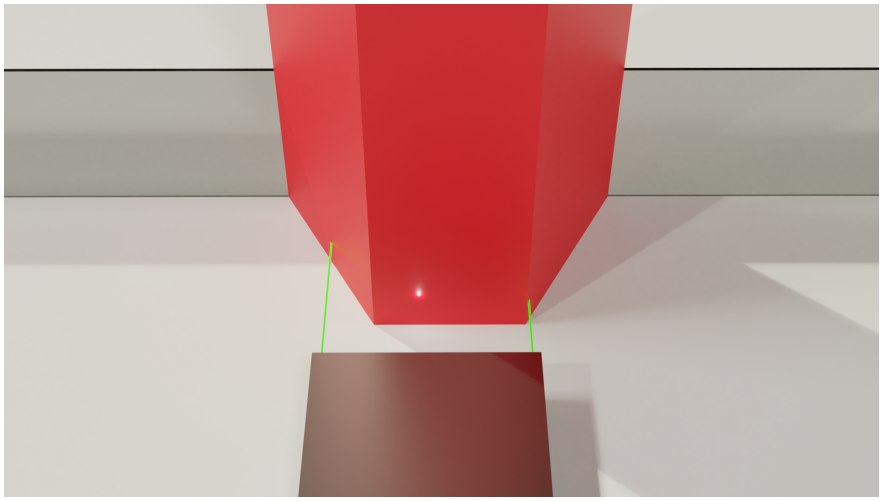

Figure 4.2: A robot to the left of the rocket with the beams of the two distance sensors shown

Using this, we ran a PID loop on the difference of these two distance readings to align the robot in the y direction. However, first we used two subsequent PID loops using an IMU and the minimum of the two distance readings to make sure the robot was facing the rocket and was the correct distance away from the rocket before aligning in the y direction. This algorithm was decently robust and worked as long as one of the distance

sensors was on the face of the rocket. A version of this algorithm before the PID loops were optimized can be seen in the following video. https://imgur.com/a/ZOy0vMM

4.3 Aligning with the FRC Hatch Port

Aligning to place Hatch Panels anywhere on the field was a more complicated algorithm. Because we had been using the TeraRanger Multiflex for the other alignment algorithms and already had a string of them on the robot, we decided to try to use them for this use case as well. Our robot had the cargo mechanism and the hatch panel mechanism on different sides of the robot, and we used two of the eight TeraRanger sensors on the cargo side. This meant that we only had six distance sensors to work with to stick with just one TeraRanger Multiflex on the robot.

The algorithm we developed constisted of two main parts. The first part is making sure the robot is facing the rocket and is the correct distance away. Then once the robot is close enough to the rocket, we figured out which sensors were looking at the face of the rocket and which were either looking through the port or off to one of the sides of the rocket. Then with this binary interpretation of the distance values we defined all of the possible cases as the robot moved side to side in front of the rocket. Then we ran a slightly refined version of bang bang control where we differentiated between cases far to the left or right of the rocket, cases just to the left or right of the rocket, and the couple cases in the center that are all close enough to place a hatch panel.

This algorithm gave pretty decent results under ideal circumstances, but we ran into a lot of issues getting it to work reliably. The biggest issue stemmed from us trying to make the algorithm work all the way from the most extreme case where only one sensor was on the face of the rocket and all of the others were either off to the left or right. Doing this instead of just trying to make the algorithm correct for smaller offsets in alignment introduced a lot of duplicate cases where it wasn't possible to tell whether the robot was just slightly to the left or really far to the right of the rocket. Additionally, trying to account for such large alignment variations caused a lot of headaches when we tried to determine all the cases when aligning to place a hatch on the cargo ship in addition to the rocket where even more duplicate cases were created. Typically if the algorithm started in a known state it would be able to ignore the duplicate cases and successfully align, but if it ever started in an unknown state or wrongfully determined the state it was in due to the rotation being

off or any other reason it wouldn't be able to align quickly or reliably. Another reason that the algorithm was unreliable was due to our attempts to overoptimize its speed. We tried to make it align in both the x and y directions at the same time as well which effectively amplified any uncertainties that could cause the algorithm to think it was in the wrong place. A few videos showing some variations on this alignment algorithm can be viewed below.

https://youtu.be/c6i4WnYGES4

https://youtu.be/2evFf09b2M0

4.4 Aligning with the TeraRanger Evo 64px

During the competition season, we didn't have enough time to experiment with alignment algorithms using the Evo. However if we had tried to develop a similar algorithm as described in section 4.3 using the Evo, it would have likely been a whole lot more robust as we would have been able to detect the rough shape of the port itself instead of just the notch at the bottom of it. We would have been able to construct a more intelligent algorithm than determining every single possible case when aligning to place a hatch panel. Additionally we wouldn't have had to deal with the headache of placing and protecting six distance sensors along a side of the robot where we hadn't planned for any sensors at all. Overall, the Evo would have given us a lot more information about both the location of the circular port and the notch at the bottom which would have allowed us to develop an alignment algorithm without enumerating through dozens of cases.

5 Future Plans

This year we will be able to devote a lot more time towards developing more sophisticated algorithms using these sensors, and since we have the sensors before build season we will be able to account for the sensors while designing the robot which will give the programming team much more flexibility. We are super excited to see how we can use these sensors in next year's game.