Benchmarking of 2D-Slam Algorithms

A Validation for the TETRA project Ad Usum Navigantium

Peter Aerts, Eric Demeester

ACRO RESEARCH GROUP, KU LEUVEN, DEPARTMENT OF MECHANICAL ENGINEERING, CAMPUS DIEPENBEEK, WETENSCHAPSPARK 27, 3590 DIEPENBEEK, BELGIUM

Juli 17, 2017

http://www.acro.be/adusumnavigantium/

Benchmark

The reason for this document is to give the reader an overview of several SLAM (Simultanious Localization And Mapping) algorithms and their performances. It's important that the generated maps are evaluated in order to get a perception of the differences between approaches.
We also hope to provide the reader with the necessary information to be able to determine and select a SLAM algorithm suitable for his or her application. Furthermore we will address some difficulties that were experienced during this evaluation and define influences that have an effect on the performances of SLAM algorithms.
This document will lead to the selection of SLAM algorithms to be used in cases chosen for the TETRA-project Ad Usum Navigantium.

For the evaluation of SLAM algorithms, the generated maps are compared to each other and not to a ground truth. This means that we do not evaluate the absolute accuracy of the maps. The goal of this evaluation is to discover strengths and weaknesses of each generated map based on each other.
Several evaluations are already conducted and published on a number of SLAM algorithms. For our evaluation, metrics were selected from litirature. We will describe these approaches and present our results further in this document.

Two main papers (Filatov et al. 2017) and (Santos et al. 2013) were consulted for our evaluation. In total there are four main evaluation criteria. The first three allow us to evaluate the certainty and accuracy of the maps relative to each other. The fourth and last approach gives us a comparison on the similarity of the generated maps.

SLAM algorithms

The chosen 2D SLAM algorithms for this evaluation are open source. The majority of these approaches are particle based except for one (Cartographer) which is a graph-based approach. Most algorithms are ROS[1] compatible. The selected and evaluated algorithms are:

[1]Robot Operating System

- Gmapping
- HectorSLAM
- TinySLAM
- Cartographer
- DP-SLAM

As previously mentioned, most algorithms are particle filter based algorithms. Every SLAM algorithm expects certain data to generate its map. Most commonly laser scan data and odometry. This is true for all evaluated algorithms except for HectorSLAM.
HectorSLAM is a particle filter based algorithm that only uses laser scan data to generate its map. This has its advantages and its disadvantages explained further in this document.

Laser scan data are distance measurements taken over a certain angle of the laser scanner. Odometry is the displacement of the wheels of the robot. This is used to generate an x ,y, theta estimation of the robot.
An other element required by the SLAM algorithms are the location of the laser scanner relative to the pivot point of the wheeled robot. In ROS, this is provided by an other data message called the transform message.

Mapping environment

The constructed maps must be generated in a realistic environment. The indoor environment chosen, was the upper floor of the Techonologie Centrum at Diepenbeek. This environment however brings with it several difficulties for the laser scanner and the SLAM algorithms.

The generated maps are occupancy grid maps. These maps are visualized as a grid of pixels. Every pixel represents a little area within the environment. The darker the pixel, the more certain the algorithm is that this area within the environment is occupied. If a pixel is brighter and approaches a more white than gray color, than it assumes this area is free space.

Long hallway

SLAM algorithms use scan matching in order to minimize the uncertainty of the odometry. The more obstacles are present within the environment, the easier scan matching becomes and the easier a linear or angular change in the robots position is detected.
The mapped environment has a hallway of approximately 60 meters. This is an obstacle for

SLAM algorithms due to the fact that, during the data collection, all laser scan data is very similar. The lack of obstacles and change in the environment makes a hallway very difficult for algorithms to perform a correct scanmatching.
Therefor a long hallway is an obstacle. Figure 1 depicts the hallway of the upper floor which

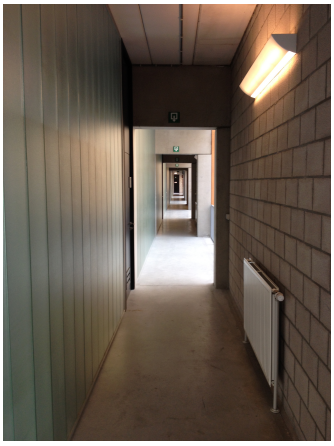


Figure 1: Hallway of the upper floor which is an obstacle for SLAM algorithms.

is present in the mapped environment.

Glass walls

Another difficult element in this environment are the glass walls. These walls are located on one side of the long hallway. This is an obstacle for the laser scanner and sequential a difficulty for SLAM algorithms. Glass can reflect the laser beam which therefore returns an inaccurate measurement. The laser beam can also pass trough the glass and return a distance measurement to the obstacle behind the glass. This makes detecting clear glass very difficult. In our environment, the glass wall consist of ground glass 2. The laser beams

Figure 2: Glass wall located in the hallway of the upper floor which is an obstacle for SLAM algorithms.

will not as easely pass trough this type of glass, but reflections can be a possible issue.

An other element within the glass wall is the support structure. A five centimeter glass structure is located within the wall. This can be considered as an obstacle right behind the wall since some laser beams can pass through the glass and detect the structure.

Railing

The last major difficulty of this environment is the presence of railings.
Railings can also form a problem due to the fact that the distance between an open space (the space between the railing) and occupied space (the space where a piece of the railing is located) is small. When approaching the railing from a distance, the laser beams see

Figure 3: railing located in the hallway of the upper floor which is an obstacle for SLAM algorithms.

an obstacle and not the open space between the metal bars of the railing (figure 3). The opening is visible to the laser scanner only when it is a position along the railing.
This creates a difficulty for the mapping process. The SLAM algorithms will first assume that the area where the railing is located is a wall while approaching. When the laser scanner moves along the railing, the open spaces are visible. This creates conflicting data and has its effect on the certainty on this area of the hallway.

Ghost measurements

An other inconvenient phenomenon are ghost measurements. This is a phenomenon which generates incorrect distance measurements due to the physical properties of the laser beams. A laser beam of a laser scanner isn't infinitely small. It has a certain width and converges wider the greater the distance. When a laser beam hits the corner of a wall as depicted in figure 4, half of the laser beam will hit the wall. The other half will pass alongside the wall and hit the next obstacle.
So there are two measurements of one laser beam which is not possible. The laser scanner will create a 'ghost measurement' located between the distance of the corner of the wall and the next obstacle.
This can be problematic for SLAM because it could translate this measurement to an occupied space within the environment while this space is actually free.

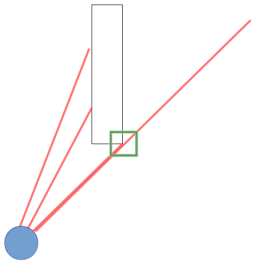


Figure 4: Situation where a ghost measurement is created.

Recording data

In order to properly evaluate the chosen SLAM algorithms, the laser data and odometry should be exactly the same for each algorithm. Therefore the data was recorded and afterwards implemented in the algorithms to create their respective maps.

Figure 5: Hallway of the upper floor which is an obstacle for SLAM algorithms.

An electric wheelchair(figure 5) was available to us to record the odometry. Both wheels have incremental encorders attached to it. These encorders will return the angular displacement of each wheel. The timestamp of these measurements are also returned and the data is written to a .txt-file. To gain laser scan data, two laser scanners where attached to the front of the wheelchair:

- Leuze RSL430-S

  max distance: 50 m

  recording frequency: 2,5 Hz

- Hokuyo URG-04LX-UG01

  max distance: 5,6 m

  recording frequency: 10 Hz

Take note that the Leuze laser scanner is in fact a safety laser scanner while the Hokuyo is a cheaper laser scanner for small robotic applications. There is a certain difference between the amount of noise on the measurements. The Leuze scanner has less noise and less zero-values than the Hokuyo laser scanner. The effect between these two scanners will become more clear further in this document.

The measurements and their respective timestamps of these two laser scanners are written to separate .txt-files. We will not go into the synchronization of the laser data and the odometry. This will be addressed in an other document.

Implementing data to SLAM

As previously mentioned, most SLAM algorithms are available in ROS. Therefore, the data must be published as ROS topics. This will allow the SLAM algorithms to subscribe to these topics and use the information to create a map. The algorithms available in ROS are:

- Gmapping
- HectorSLAM
- TinySLAM
- Cartographer

All these algorithms use odometry data and laser scan data except for HectorSLAM, which uses only scan data.
To publish the data via ROS a node was created. This node reads in all separate files and publishes an odometry message, a message containing all laser scan data and a message containing all transforms between the laser scanner and base of the robot and the starting point of the robot.

DP-SLAM uses a .log-file to run offline. This file consists of several lines of data where the primary line is the robots position and pose and the secondary line the laser data. To generate this file, a Python script was created which reads in the odometry text file and the laser data text file. It converts the displacement of the wheels to the robots position and orientation and generates the necessary .log-file.
DP-SLAM assumes that the odometry and the laser data are written in the file at the same time. Therefore a linear interpolation was conducted in order to get an accurate estimate of the x,y and theta of the robot on the timestamp of the laser data.

Adaptations to SLAM

In order to correctly evaluate each map, they need to be similar in their construction. Therefore some algorithms were adjusted.

Output occupancy grid map

For instance, Gmapping and HectorSLAM each use a threshold on the maps probability. Every cell/pixel has a certain value between 0 and 100 were 0 is an absolute certainty of occupation and 100 is an absolute certainty of free space. The value -1 is considered as unknown space.
HectorSLAM and Gmapping use thresholds to only define absolute occupied and absolute free spaces. They do not return values between 0 and 100.

Eventually we changed the output of the algorithms so that the output ignores the tresholds and returns all probabilities generated by the algorithm itself. The unknown space is also remapped to a value of 50. This means that the algorithm doesn't know if this space is free or occupied. There is a 50/50 chance between occupied or free space.

In order to save these maps and all their probabilities, a new package in ROS was created to write all values to a .pgm file. In a later stage these .pgm files are converted to a .png file without any loss of data.

TinySLAM

TinySLAM in contrary to Gmapping and HectorSLAM returns all probability values of its map. However some problems have arisen during the configuration and implementation of TinySLAM.
First of, the tutorial available at the ROS wiki page is very limited in its explanation of the algorithm. The algorithm doesn't use the standard naming of topics to subscribe to. TinySLAM was originally listening for a topic named *laser_ scan* for laser data. The data however was published on the topic name *scan*. The same applies to the odometry frame which was expected by TinySLAM to be named *odom-combined* where as the general name is *odom*.
These names were changed in order to quickly and easily test our own data on all algorithms.

Another problem with TinySLAM was the ability to save the published map. First of all, a new package was necessary in order to save all probability values within the map. This was also mentioned in the section above. Another problem was the fact that the map wasn't published repeatedly but only during an update of the map. This means that when all data was used and the map generated, the final map wasn't repeatedly published to ROS. The created package to save the map waits for the map to be published into ROS and then constructs a .pgm file. Therefore we needed to change the code so that the map was constantly published.

Parameters

An other element which needed to be changed where the parameters used for each algorithm. These algorithms don't work out of the box, some parameters needed to be adjusted in order

to get the algorithm working properly. Most parameters were held the same for different algorithms if this was possible.

HectorSLAM only needs laser scan data. It uses scanmatching in order to fit its current scan to the previous scan in order to determine the robots movement.
The maps moment of update is defined by the linear or angular movement of the robot. HectorSLAM has build in parameters set on distance and angular displacement to define when the map needs to be updated. Other parameters such as the map size, number of particles and names of ROS topics can also be adjusted. The following parameters where adjusted to receive a good representation of the environment using HectorSLAM:

- map size
- linear update
- angular update
- maximum distance of laser scans
- number of particles

The map size parameter represents the size of the generated map in pixels. This was set to 4000 which creates a maximum map size of 4000 x 4000 pixels.
The number of particles was also adjusted. This parameter was set to 50 instead of the standard 30 number of particles. This is done to create a more accurate map. We could set it higher but this will increase the computational effort needed by the algorithm.
The maximum distance which the laser scanner can measure was also adjusted. This was set to 5.6 meters for the Hokuyo laser scanner and 50 meters for the Leuze laser scanner.
Last, the linear and angular update was adjusted. This was done so that HectorSLAM would update its map quicker which provided better results. The linear update was set to 1 mm and the angular update was set to 0.05 radians.

Gmappings parameters that can be changed, strongly resembles those of HectorSLAM. For our case, the changed parameters are:

- linear update
- angular update
- number of particles

Because we want to evaluate the SLAM algorithms at an equal performance, we have set the parameters to the same values as those of HectorSLAM.

Cartographer has an individual configuration and launch file. The configuration file also containes information such as maximum laser distance, angular and linear update etc.
The main difference compared to Gmapping and HectorSLAM is the use of an IMU. Gmapping relies on the transformation message between the starting point and the base of the robot being send via ROS to get the current position of the robot. Cartographer is designed to work with IMU data but can also work with odometry. Therefore an odometry message was created using the data recorded from the wheelchair. The use of IMU data was excluded by setting this parameter to 'false'.
Cartographer has a wide range of parameters that can be set. Again the linear and angular update was set to the same values as HectorSLAM and Gmapping. Also the maximum laser distance was provided in the configuration file. Other parameters such as sample ratios and publish periods were kept at their default value.
The launch file calls the cartographer node. This node needs to know which configuration file it needs to use which is also defined in the launchfile. Take note that we used the offline node from cartographer and not the node for real time mapping. This was done in order to speed up the simulation time.

Generated maps

Eventually every SLAM algorithm was tuned and has used exactly the same data.
Two sets of maps were constructed. Figure 6 displays the maps generated with the Hokuyo laser data set. Figure 7 displays the maps generated with the Leuze laser scanner.

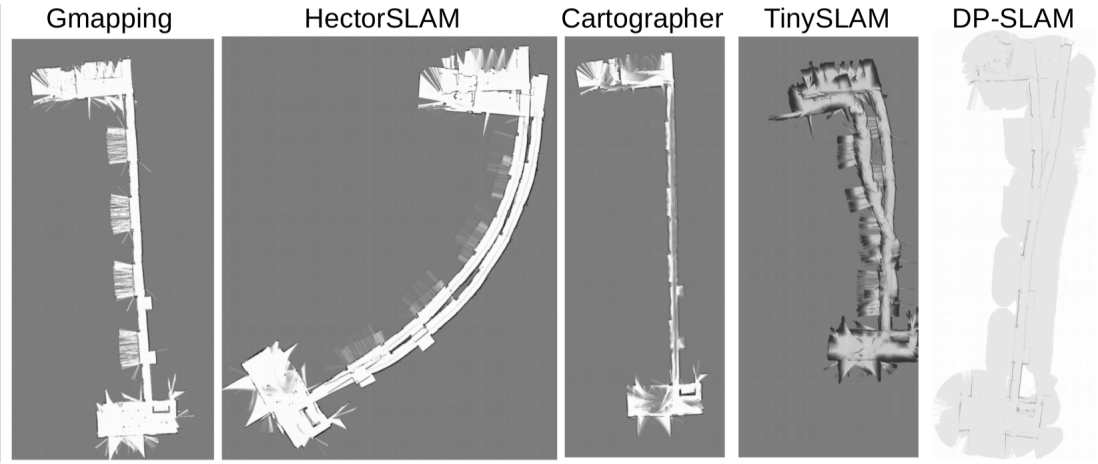


Figure 6: Maps created using the various algorithms and the Hokuyo laser scanner.

We encountered problems getting DP-SLAM operational. The algorithm works with the provided dataset but would not run with our own. The algorithm does start the creation of a map, but halfway it runs into a segmentation fault.
While searching to solve this we were able to overwrite the a specific part of the memory which got DP-SLAM operational with the data set of the Hokuyo laser scanner. However

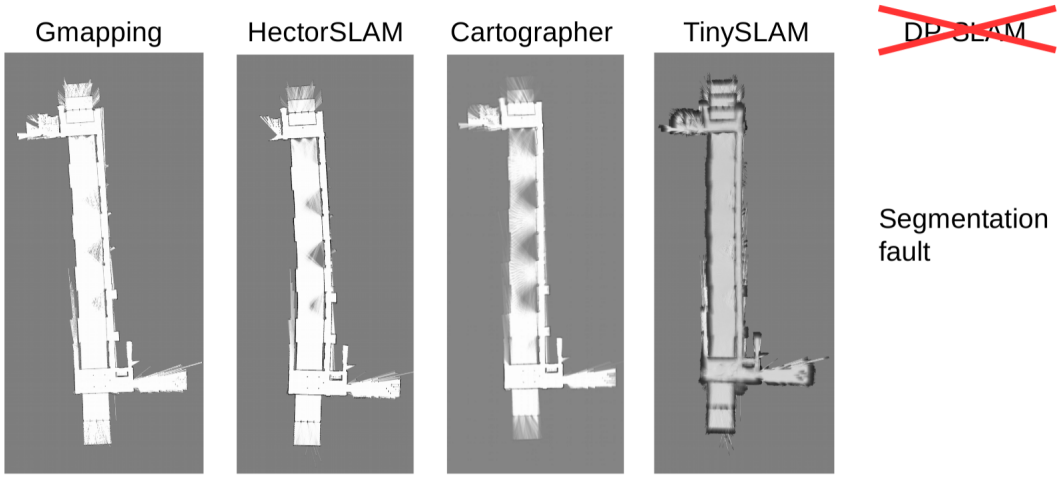


Figure 7: Maps created using the various algorithms and the Hokuyo laser scanner.

this 'fix' was not able to solve the segmentation fault for the Leuze data set. After some time searching to solve this segmentation fault, we decided to omit DP-SLAM and focus our efforts on the more userfriendly algorithms. Take note that DP-SLAM is known for its segmentation faults. The latest version (which we used) already solved some of these faults.

We selected four metrics to use for our benchmark. These metrics were applied to the maps generated with the Leuze laser scanner data. This choise was made as a result of the visual inspection.

Visual inspection

The first evaluation was done by visual inspection. This is recommended in literature to be a general means for evaluation.
When we look at the respective maps generated with the Hokuyo laser scanner and the Leuze laser scanner, we can already visually see that the maps with the Leuze laser scanner are better. For instance, when we look at figure 8, we can see that the dataset of the Leuze laser scanner creates much straighter lines than the Hokuyo data set. With this evaluation we can account for our choice to use the maps generated with the Leuze data set for the remainder of the benchmark.

An other visual property is the thickness of the generated lines within the map. The lines represent the walls within the environment. However, if the lines are thicker, the algorithm is more uncertain of the exact location of the walls. In a perfect map, the walls are represented by a single black line of pixels.
By examining figure 9, we can see that the lines generated with the Hokuyo data set are thicker than the lines generated with the Leuze data set. This differance is visable in every

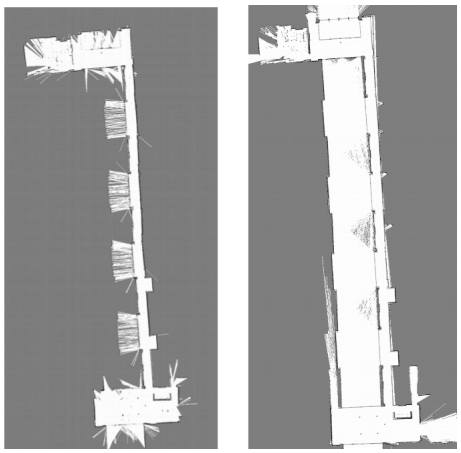


Figure 8: Gmapping with both data sets. On the left the Hokuyo data set wich creates a small curve in the long hall. Right the Leuze data set with wall that are more straight.

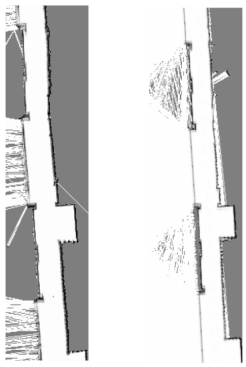


Figure 9: Gmapping with both data sets. On the left the Hokuyo data set which has much thicker lines than the Leuze data set.

tested algorithm except for TinySLAM due to the fact that the lines generated by TinySLAM are generally very thick.

An other conclusion of the visual inspection is that Gmapping and Cartographer are more robust. These algorithms can handle noisy data much better and exhibit much less drift. As visable in figure 10, Cartographer and Gmapping create decent maps. While TinySLAM creates two hallways and HectorSLAM creates a large curve instead of a straight hallway. This is due to the noise of the Hokuyo laser scanner. The laser data isn't as accurate and returns many zero-values. The range of the Hokuyo laser is also shorter than the range of the Leuze laser scanner. These two factors play a great part in the amount of drift and accurate map building.

As previously mentioned, the thickness of the lines representing a wall is an evaluation criteria for these algorithms. When all maps are viewed, we can clearly see that TinySLAM

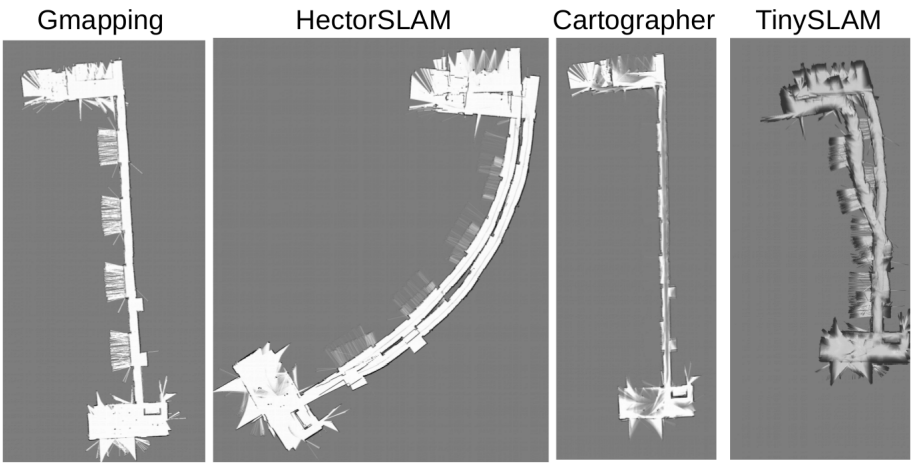


Figure 10: Gmapping and Cartographer build maps which represent the actual environment.

generates the thickest lines within the map. This means that this algorithm is the most uncertain of its environment and does not know the location of the walls as precise as the other alogorithms.
Figure 11 shows the TinySLAM map compared to the Cartographer map.

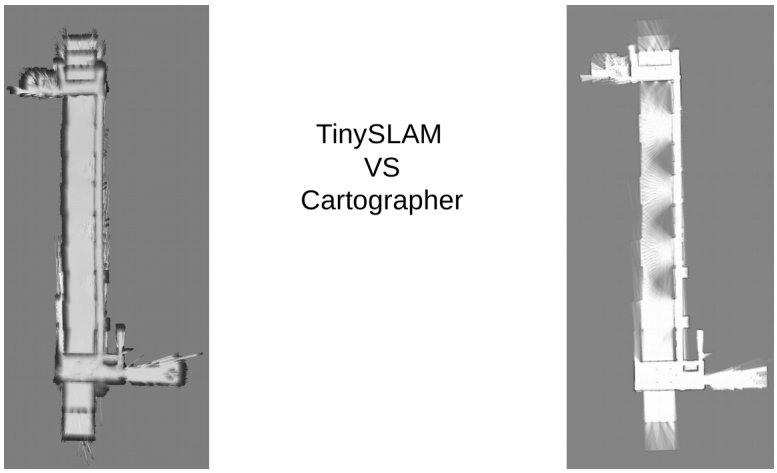


Figure 11: TinySLAM compared to Cartographer. The generated lines of TinySLAM are thicker than any other algorithm meaning that it is the most uncertain of the location of the walls in the environment.

A last visual conclusion is the importance of sensor fusion. When looking at HectorSLAM with both data sets, we see that the hallway is curved. This is due to the fact that HectorSLAM only uses laser scan data. The scan matching HectorSLAM performs can not account for the drift. Other SLAM algorithms can account for this drift due to their sensor fusion. The combination of odometry and laser data creates very straight hallways when

the laser data isn't to noisy.Figure 12 represents the two maps generated by HectorSLAM. We can clearly see that the hallway isn't straight. Due to the fact that the Leuze dataset

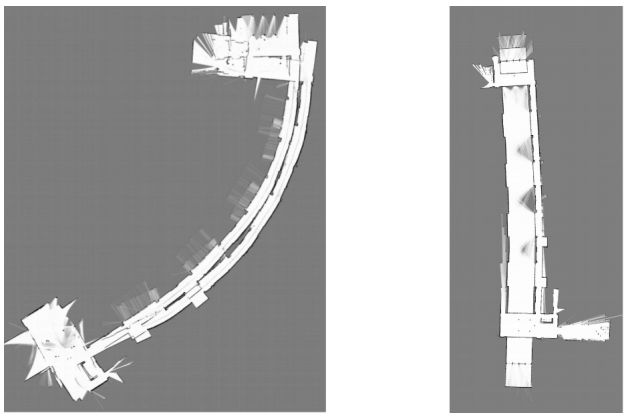


Figure 12: HectorSLAM: on the left the map with the Hokuyo dataset, on the right the Leuze dataset.

contains much less noise, HectorSLAM does create a good representation of the environment. For environments where enough diversity is present, HectorSLAM can perform admerably. However for the hallway problem, HectorSLAM does have some difficulty with generating a straight hallway.

Conclusion

As a result of the visual inspection we can conclude that the dataset recorded with the Leuze laser scanner results in better maps than the dataset of the Hokuyo laser scanner.
For further evaluation, these maps generated with the Leuze dataset where used.

TinySLAM, with its small size of 200 lines of code, creates the thickest lines. In other words, it does not define the position of walls as accurate as the other algorithms. Therefore its the most uncertain algorithm.

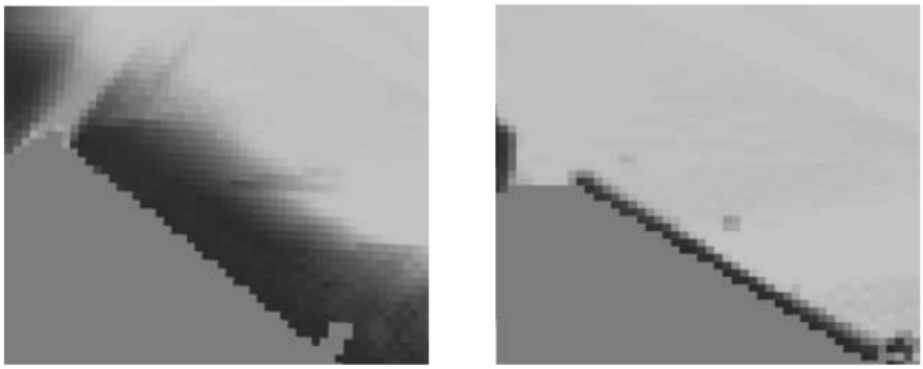
Sensor fusion is an important factor. When comparing HectorSLAM, it is visible that the hallway is a less accurate representation in comparison to the other algorithms. This once more underlines the importance of sensor fusion. However, HectorSLAM can preform admirably in small office spaces.

Due to DP-SLAMs segmentation fault, this algorithm is not considered further in this project. The effort needed to get this algorithm operational within this projects time frame is unrealistic.

From this inspection the conclusion is made that Gmapping and Cartographer are the most promising algorithms.

Proportion of occupied space

As a first metric within this benchmarking, the proportion of occupied space was calculated and compared. This metric is described in a paper byFilatov et al. 2017.
The proportion is a measurement of how certain the algorithm is of the location of objects and walls. The darker the pixel of the map, the more certain the algorithm is of the object or walls location. With two maps that are similar, the more blurriness in a map, the lower the quality. Figure 13 represents the same wall. The left figure has more blur compared to the right figure. Therefore the left figure is less certain of the position of its walls.



(a) With blurry effect (b) Without blurry effect

Figure 13: HectorSLAM: on the left the map with the Hokuyo dataset, on the right the Leuze dataset.

Calculation

In order to get the proportion of each map, a script was written in Python. In this script the total number of pixels considered occupied is calculated. A white pixel represents free space which corresponds to a value of 255 in an 8-bit image. A pixel considered as occupied space corresponds to the value 0. All pixels below the value 127 are considered occupied. By counting all these pixels we calculate the number of occupied pixels.
Take note that only the number of occupied cells/pixels isn't enough to evaluate these maps. For if we compare maps based on this number, we overlook the possibility that some algorithms generate correct and complete walls where an other will not. Comparing them using the total number of occupied cells is only justified when the corresponding maps fully represent the total environment. This is not guaranteed, therefore we calculate the proportion.

As the second step, the Python script converts each map into a binary image using a threshold from the occupied cells. This results in a map only defining white and black pixels. This corresponds to free and occupied space. This threshold is the mean of only the occupied cells. After the binarization, all black pixels i.e. occupied spaces are counted.

This number is devided by the total amount of occupied cells concidered within the map. This gives us the proportion of each map.

Results

The following table 1 shows the results for all the maps generated with the Leuze dataset. The

	Gmapping	HectorSLAM	TinySLAM	Cartographer
Occupied cells	12053	22386	95244	15012
Mean	68	60	84	100
Occupied cells below Mean	5467	10241	49443	5033
Proportion	0.45358	0.45784	0.51912	0.33527

Table 1: Results of proportion calculation

higher the proportion, the more blurry the map represents its walls. From these calculated values we can see that TinySLAM has the highest proportion, followed by HectorSLAM and Gmapping which are very close together. Cartographer has the lowest proportion value. This means that Cartographer generates the finest lines representing walls.

Conclusion

From this first metric we can conclude that TinySLAM is the least promising algorithm. Its proportion value can be linked to the visual evaluation. We concluded that TinySLAM is the most uncertain of the location of its walls because it generates the thickest lines. With this metric, our visual inspection is confirmed.
An other conclusion which can be made is that HectorSLAM and Gmappinng are similar in their value. These algorithms create their walls with almost equal certainty.
Considering this metric, Cartographer is the most promising algorithm. With its very low proportion its the most certain of the location of its walls and generates the finest lines.

Number of enclosed areas

The second metric for evaluating SLAM generated maps is the number of enclosed areas. This metric is primarily used to distinguish between low-quality and high-quality maps. We define an enclosed area as an area of occupied space fully surrounded by free space and visa versa.
This metric will identify which algorithm has the most mismatches in trajectory and most ghost points considered as occupied space.

Calculation

To get the number of enclosed areas of each map, a Python script was created which first converts all undefined space to occupied space.
Every pixel with a value of 127 in an 8 bit image is converted to the value of 0.

The next step in the calculation of the number of enclosed areas is to transform the 8-bit map to a binary map. This was done using Otsu's binarization method. This method calculates a treshold based on the histogram of an image. In short, Otsu's binarization method defines the threshold value as the mean of the two largest peaks within the histogram of an image. This function is available in OpenCV which is an open source library for image manipulation available in Python and C++.

Once we have acquired a binary image of the map, Suzuki's algorithm Suzuki, S.et al.,1985 is applied to find all enclosed areas. The algorithm is available in OpenCV and returns an object list. Each object consists of vector points describing each contour of the image. Figure 14 is the visualization of all contours on each map.
The number of objects within the list are counted which returns in the total amount of

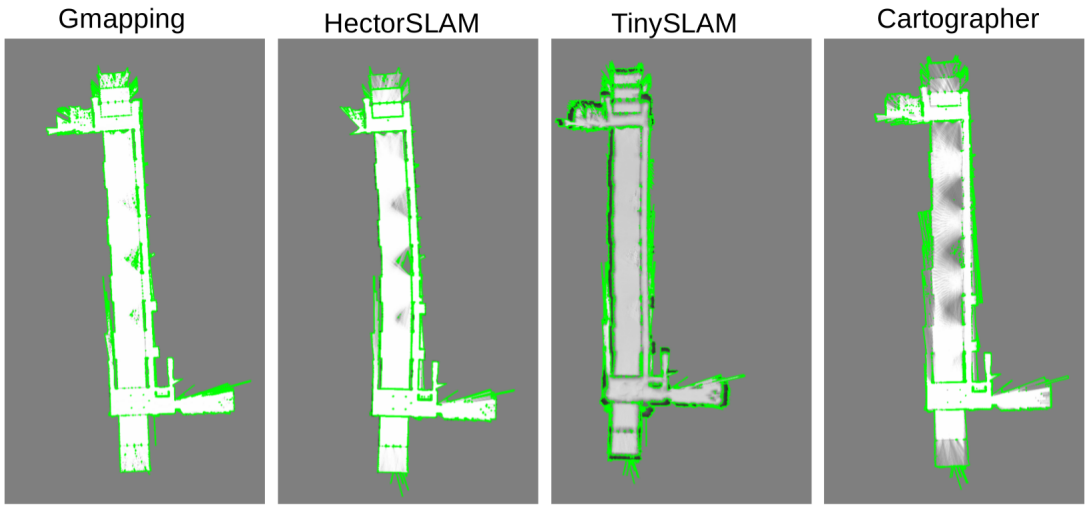


Figure 14: All enclosed areas visualized on each map.

contours within each map.

Results

Table 2 shows the results of the previously mentioned metric. The results show that HectorSLAM has the least amount of enclosed areas. This can be allocated to the fact that HectorSLAM only uses laser data and scan matching. therefore all scans are better aligned and do not generate extra enclosed areas.

	Gmapping	HectorSLAM	TinySLAM	Cartographer
Otsu's Treshold	124	120	92	104
contours	1346	521	2288	2078

Table 2: Results of the number of enclosed areas

TinySLAM has the most enclosed areas. This means that TinySLAM again proves to create maps of lesser quality compared to other SLAM algorithms.

Conclusion

The results show that HectorSLAM creates the least amount of contours and therefore is the most favorable algorithm. TinySLAM on the other hand is the least favorable algorithm because it generates the most enclosed areas of all four algorithms. Gmapping is second best and Cartographer is third in line.
Some elements must be addressed when examining these results. First, ghost measurements have a large influence on this metric. Also Cartographer creates submaps and aligns these maps to create a global map. When these submaps are slightly misaligned, new enclosed areas can be created.
Nevertheless, this metric places TinySLAM last in line and HectorSLAM seems to create the least amount of enclosed areas.

Corner count

This third metric for comparing the generated maps is to count the amount of corners within the map. For maps of similar quality generated by SLAM algorithms on the same dataset, the map with the least amount of corners is more likely to be more accurate and more consistant.
Extra corners are errors which can occur due to trajectory mismatch. This mismatch can lead to extra walls or additional features or objects within the map which are not physically present within the environment. This will create extra corners within the map.
Data noise and ghost points can also be able to create extra corners within the map.

However, this metric is not suitable for TinySLAM. The map generated with this algorithm does not resemble the other generated maps. The wide lines representing walls has as effect that the approach of the corner count isn't applicable.

Calculation

This metric does not calculate the exact amount of corners within the map. The approach generates an array with all probabilities of areas within the map being a corner. The number of areas above a certain threshold is considered as the metric for the benchmarking.

The first step within this process is to map all unknown values of the maps image to occupied values. This means that in an 8 bit image of the map, all pixels with value 127 are set to 0. After that, a Laplace-Gaussian filter is applied to obtain an abstraction of the maps image. The larger the sigma chosen for the filter, the wider the lines of the abstraction become. Its due to this step that TinySLAM can't be used for this metric. The abstraction of TinySLAMs map creates straighter lines due to the thick lines of TinySLAM. The filter has a wide tolerance range to generate its abstraction. Therefore the abstraction doesn't resemble the abstraction of the other maps.
This Laplace-Gaussian filter is available in Scipy which is a scientific library created for Python.

The abstraction of the map is implemented in the Harris corner detector. The original paper Harris, C., et al. (1988) explains the approach for detecting corners. This function is available in the open source library of OpenCV. This algorithm returns an array of calculated scores to determine the possibility of a corner.
We implemented a certain threshold and counted the number of probabilities above this threshold for comparison of the maps.

Results

Figure 15 is a visualization of the located probabilities above a certain threshold. The red dots within the map are then counted and used as a metric for determining which SLAM algorithm proves to generate the least amount of possible corners. The corner count was

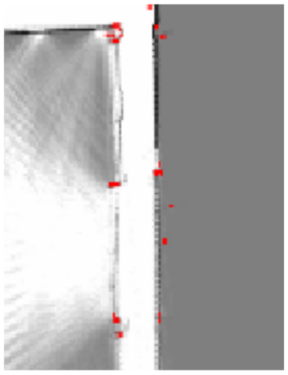


Figure 15: The red dots represent possible corners which are above a certain treshold. This map is a piece of the hallway generated with Cartographer.

executed three times. The first calculation was done with the sigma parameter for the Laplace-Gaussian filter set to 5 and the threshold set to 20% of the maximum returned value of the Harris corner detector. Figure 16 shows the map after the application of the filter. The second evaluation was executed with the same threshold. The sigma parameter for the filter was set to 3.

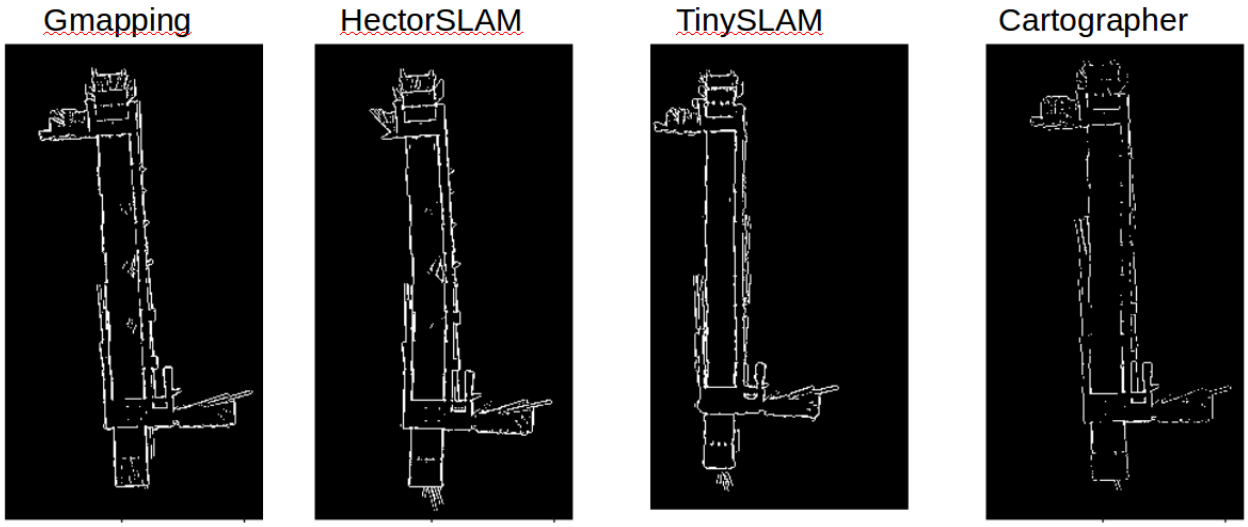


Figure 16: These four maps are the result of the Laplace-Gaussian filter with a sigma of 3. As visable, the filter generates much less corners and more straight lines. This is due to the fact that TinySLAM generates very thick lines within its map.

		Gmapping	HectorSLAM	Cartographer
L-G sigma/Treshold	5/20	509	635	209
L-G sigma/Treshold	3/20	687	1226	388
L-G sigma/Treshold	3/10	254	509	127

Table 3: Results of the corner count metric. Take note that the result isn't the actual number of corners but is the amount of possible corners within the map.

The last computation maintained the sigma of 3 and the threshold was set to 10% of the maximum value of the Harris corner detector. Table 3 shows the results of the calculation of this metric. Take note that this metric isn't applicable to TinySLAM for reasons earlier mentionned.

This table shows a consistancy in the data. By increasing the sigma of the filter, the abstract map as shown in figure 16 retains a wider representation of the walls and therefore less possible corners are detected.

Conclusion

From the results we can conclude that Cartographer is the most favorable algorithm because it generates the least possible corners in comparison with Gmapping and HectorSLAM. HectorSLAM generates the most possible corners and therefore is the least favorite algorithm. This result is consistent for different sigmas for the abstraction and different thresholds.

K-nearest neighbours

The last metric originates from Santos, et al., 2013. This comparison gives an idea of which maps mostly resemble each other.
The idea is to align two maps on top of each other and then to check the mean deviation in distance between these two maps.

Knn search

Figure 17 is a visualization of the knn-search technique. This technique calculates all distances of each data point to their respective closest neighbour. Therefore there needs to be a fixed data group, for example the red dots, and a data group to considered as neighbours, for instance the blue dots. In this example all distances calculated from the red dots to the

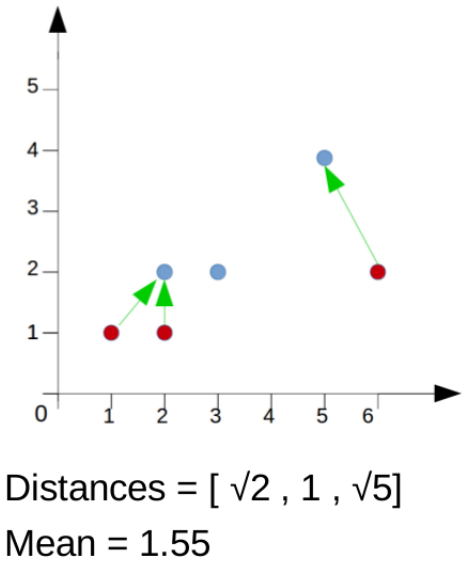


Figure 17: Red dots is the fixed data group. The blue dots is the datasec considered as the neighbours.

closest blue dots returns three values: $\sqrt{2}, 1, \sqrt{5}$. The calculated mean has a value of 1.55.

One must take into account that the lack of data points in the data group considered as neighbours has a large effect on the mean calculation. Because when every red dot searches for its closest neighbours and data points are missing, its closest neighbours are further away. Figure 18 visualizes this effect. This means that, when we apply this technique to our maps, the lack of data points has a large influence in comparing the resemblance of the maps.

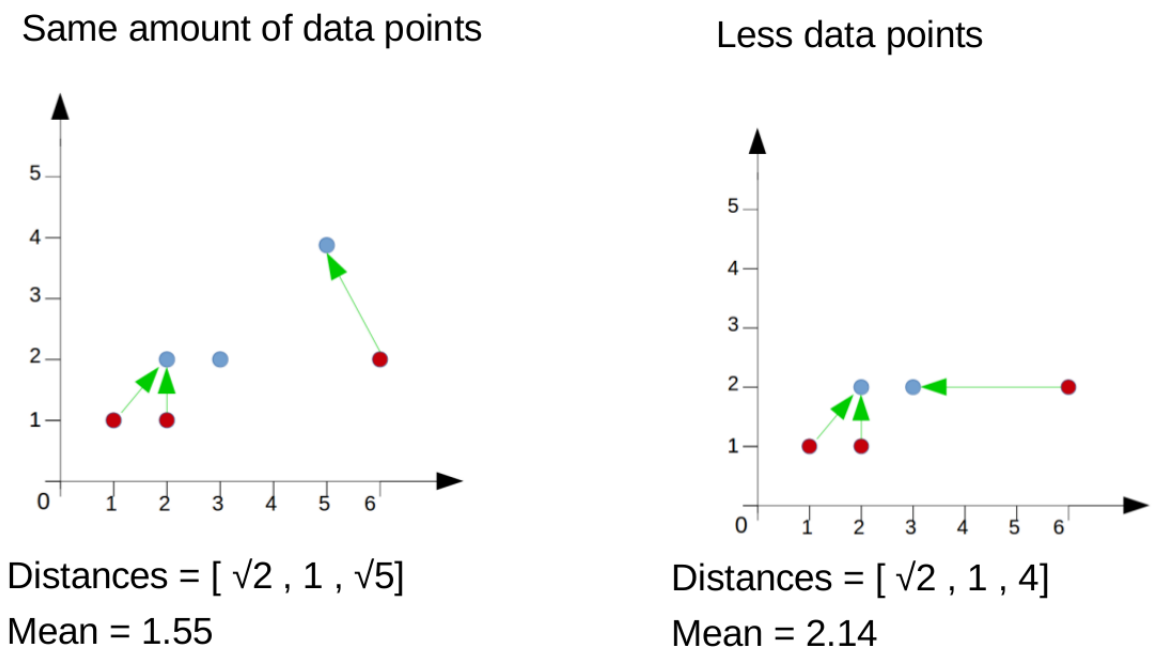


Figure 18: On the left, the same amount of neighbour data points. On the right the upper right datapoint is missing. This leads to a higher mean calculation.

Calculation

The first step in the calculation of the mean deviation between the data points is a binarization step. This was done using Otsu's binarization method. This results in two maps which are binarized but not yet aligned. Figure 19 shows two maps on top of each other without any translation or rotation. In order to apply the knn-search technique, an alignment of

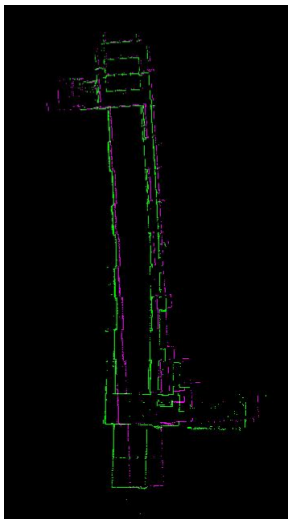


Figure 19: Two maps, Gmapping (purple) and HectorSLAM (green), which are not aligned.

the two maps must occur. The Matlab toolbox has a function to execute a transform of an image to align two images.
The function *imregister* within the matlab toolbox transforms an image to fit another image based on intensity. This function aligns the two maps in order that the most data points (pixels considered occupied) align. Figure 20 represents the aligned maps of figure 19. When

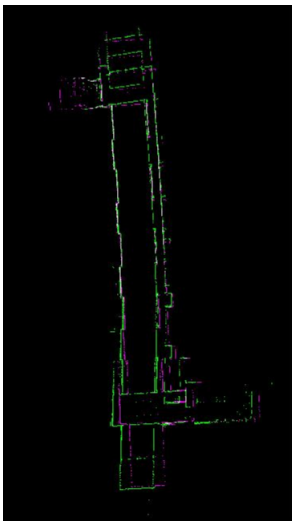


Figure 20: Two maps, Gmapping (purple) and HectorSLAM (green), which are aligned.

these two figures are aligned, the x and y position of every pixel considered occupied is written to a separate array.
These two arrays are used for the knn-search, which returns one array with all distance values. From this array, the median and mean is calculated.
Because the x and y positions are the positions of the pixels, the outcome will be the mean and median pixel distance. Take note that this approach is not applicable to TinySLAM. Due to its wide lines, almost every map can fit within the data points of TinySLAM. This means that every neighbour is found on top or very close to the data point. This leads to the misassumption that TinySLAM resembles all maps almost equally. Figure 21 visualizes this problem.

Results

The results are shown in three tables 4. Every table has an other map chosen as base map. Within each table the mean and median are presented for each other map. Take note that TinySLAM is not present within these tables for reasons earlier mentioned. There is a differance between the results of two maps depending on which map is the base map. As previously explained, the lack of data points has a large influence on this metric. The differance is a consequence of one map having more data points than the other.

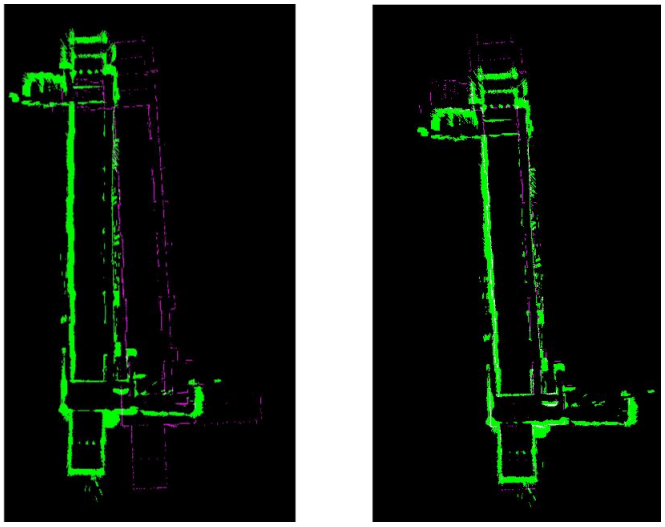


Figure 21: Gmapping (purple) and TinySLAM (green). The image on the left visualizes the two maps which are not aligned. Notice that TinySLAM has very wide lines. The right image are the two maps aligned. We can clearly see that Gmapping almost fits perfectly in the map of TinySLAM due to the wide lines.

Gmapping		
	Mean	Median
HectorSLAM	7.7477	2
Cartographer	7.8384	3

HectorSLAM		
	Mean	Median
Gmapping	12.8951	5
Cartographer	7.0914	4.1231

Cartographer		
	Mean	Median
Gmapping	33.7366	8.0623
HectorSLAM	12.362	7

Table 4: Results of the knn search. Each table represents the results for an other map chosen as base map.

Conclusion

This metric is a means to uncover which maps resemble each other the most. From the results presented in table 4, we can conclude that the maps generated by HectorSLAM and Cartographer are the most similar. Gmapping resembles HectorSLAM the most. But the differance with Cartographer is small.
TinySLAM isn't considered in this evaluation due to the fact that it does not resemble the other maps because of its thick lines generated by its uncertainty.

Overall conclusion

We have evaluated several SLAM algorithms based on a visual inspection and four metrics in order to find the most suitable algorithms for this project. We collected and presented the data concerning the accurarcy, certainty and resemblance of each map.
The selected SLAM algorithms were:

- DP-SLAM
- Gmapping
- HectorSLAM
- Cartographer
- TinySLAM

Two data sets were used. One with the Leuze laser scanner and one with the Hokuyo laser scanner. The maps generated with the Hokuyo data set were only used for the visual inspection. The four metrics were executed using the maps generated from the Leuze dataset.

DP-SLAM was not included in the evaluation. The first conclusion is that DP-SLAM is an algorithm which is not user friendly. Due to its segmentation fault, this algorithm could not be properly evaluated. The decision was made to omit DP-SLAM for the moment.

TinySLAM is an algorithm that has the least amount lines of code. The visual inspection of each generated map showed that TinySLAM created the most crude and uncertain map. This assumption was substantiated with the calculation of the proportion and number of enclosed areas. The fact that its map does not resemble the other generated maps ensured that the other metrics were not applicable.

Gmapping is one of the most robust algorithms. It succeeded in building a representable map using the Hokuyo data set. From this we can conclude that Gmapping is better in handling noisy data and short laser measurements in difficult environments.

Cartographer is the only graph-based approach used in this evaluation. From the visual inspection, it can be concluded that its a robust algorithm. Similar to Gmapping, Cartographer succeeded in generating a representable map using the Hokuyo dataset. This algorithm is the most certain of the position of objects and walls. It generates the least possible corners which means it can deal with noisy data better than other algorithms. Cartographer is one of the most promising algorithms considered for this project.

HectorSLAM is an algorithm which only uses laser measurement data. The visual inspection showed that it is not as robust as Gmapping or Cartographer. Although its map build with the Leuze dataset shows a slight curve, the metrics show that HectorSLAM is not to be discarded. It has the least amount of enclosed areas and resembles Gmapping in its certainty of occupied space. And despite of the fact that it generates the most possible corners, it resembles the map of Cartographer the most.

The overall conclusion is that Cartographer is the most promising. Gmapping and HectorSLAM are also considered for further use in this project. TinySLAM generates the least promising maps. Therefore this algorithm will still be used and further evaluated but will not be prioritized. We omit DP-SLAM for the moment due to its segmentation fault.

Insights

During the benchmarking of the SLAM algorithms, some difficulties were encountered and insights acquired.
As previously mentioned, the mapped environment has some difficult aspects. We observed that reflections are difficult for these SLAM algorithms. This is mostly due to the laser scanner. The measurements return the reflected distance to the next object. If this data isn't filtered on intensity, SLAM algorithms can not identify the measurement as a reflection. This leads to difficulties in map building.

Most of the algorithms we evaluated are freely available in ROS (Robotics Operating System). However, to work with these algorithms, a fair knowledge of ROS is needed. This was acquired over a noticeable period of time.

The choise of the SLAMs parameters is an important factor. Although we strived to maintain the default versions and the same parameters for each SLAM algorithm, some adjustments needed to be made. In most cases the update frequency needed to be higher. The number of particles was also adjusted mentionend in the section **Adaptations to SLAM**. Figure 22 shows the difference between a good set of parameters for HectorSLAM and the default parameters. The choice of parameters highly depend on the sensors used on the robot. The frequency these sensors have and the speed by which they move in their environment can demand other parameter values.
When these algorithms are used in real-time, the computing power needs to be considered. Changing parameters can add computational costs to the algorithm. When working on a live robot, the speed of the algorithm becomes a factor so that no data is missed during the map building process.

When recording data, the speed of movement and frequency of the data logging has a large influence for map building. Due to the fact that the Leuze laser data is logged at a frequency of 2,5 Hz, fast movements create misalignments and problems for scanmatching.

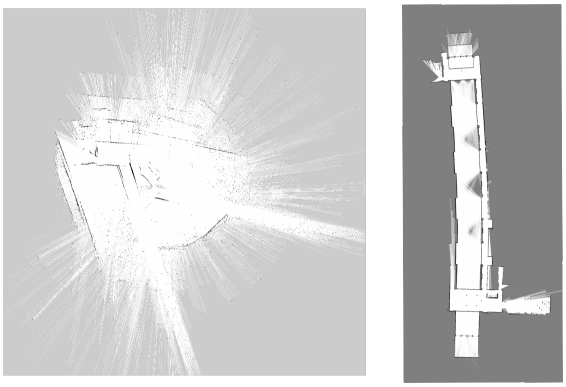


Figure 22: Two maps generated with HectorSLAM. The left map was generated with the default parameters. The right map was generated with a higher update frequency and more particles used by the particle filter.

The amount of data of the environment must be large enough for building a representable map. Therefore the recording speed is an important factor.

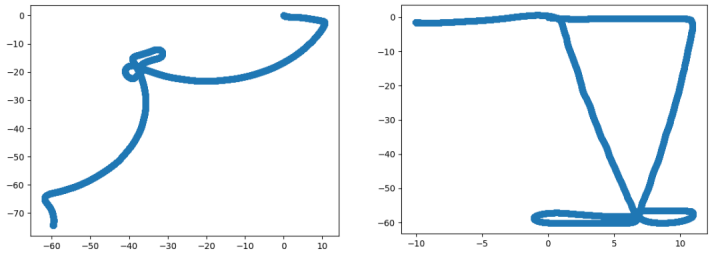


Figure 23: Path taken by the robot. The left image shows the path calculated with a wheel radius of 0.175 meters. The right image shows the path calculated with the radius of the left wheel set to 0.171 meters and the right wheel radius set to 0.174 meters.

Last, precision of the robot model is also a factor that needs to be addressed. We encountered the extend of this factor during the benchmarking. In order to incorporate the odometry into the SLAM algorithms, a conversion was made from the angular displacements of the wheels to the robots x and y-coordinates and pose. Figure 23 shows the path traveled by the robot. In the first case, the radius of the wheels was set to 0.175 meters. We can clearly see that this path does not contain straight traveled lines. This conversion to x,y,theta coordinates is therefore not accurate enough and this can form a problem for the SLAM algorithms.
The right image shows the path calculated with the radius of the left wheel set to 0.171 meters and the radius of the right wheel set to 0.174 meters. This creates a more accurate representation of the traveled path. These correct parameters show better results within the maps.

Bibliography

[1] Advances in Intelligent Systems and Computing, Reis,L.P., Moreira, A. P., Lima, P. U.,Montano, L., Muoz-Martinez, V. (2016). Robot 2015: Second Iberian Robotics conference: Advances in robotics, volume 1.

[2] 2D SLAM Quality Evaluation Methods, Filatov, A., Filatov, A., Krinkin, K., Chen, B., Molodan, D. (2017).

[3] An evaluation of 2D SLAM techniques available in Robot Operating System, Santos, J. M., Portugal, D., Rocha, R. P. (2013) IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2013.

[4] A Combined Corner and Edge Detector, Harris, C., Stephens, M. (1988).

[5] Topological structural analysis of digitized binary images by border following, Suzuki, S., be, K. ,Computer Vision, Graphics and Image Processing, 30(1), 3246 (1985).