22/08/2021

Smartly discover an environment using IoT sensors with cloud computed machine learning

MSc Robotic & Smart Technologies

by Timothee ANTHOINE-MILHOMME

Staffordshire University, Stoke-on-Trent ID: ***20020598***
Ecole Catholique des Art et Métiers, Lyon ID: ***1421 76595***

Research Supervisor: Mrs Debi Roberts

Module: MSc Project

Code: ENGG71020-2020-SPG3-2021-SPG1

Staffordshire University, Stoke-on-Trent

Abstract

The machine learning domain is gaining a lot of interest in the scientific community as well as in the industry. Various machine learning models are applied to every field where automation is possible. The autonomous vehicle industry is growing rapidly, and the exploring vehicle are developed for various application. With these vehicles being smaller and smaller the computing power required by the machine learning model is not always available on the small device. Therefore, with the rise of IoT devices it is now possible to have smart devices computing their model on a cloud server for huge computing power. In this project will be developed the prototype of an autonomous vehicle able to communicate in real time like an IoT device to a cloud computing platform. The goal of this prototype will be to map an unknown 2D environment using a 2D Lidar and a 9-axis inertial sensor. The last part of the project wasn't fully developed but theoretically analysed and schematized. The goal will be to create a reinforcement model, allowing the robot to learn how to efficiently discover the unknown environment.

Acknowledgements

I wish to thank Mrs Debi Roberts my project supervisor for her support and advice provided all along this project realisation. I also wish to thank Professor Tony Dodd who helped me for the specific SLAM question I had. I would like to thank Mr Peter Smith, at the Staffordshire FabLab, who helped me with the 3D printing parts needed for the project. I want to thank Staffordshire University and its staff without whom this project couldn't exist. Finally, I'd like to thank Mr Jan Xu from Berkley University who let me use his SLAM model and help me to adapt it to all my needs.

Table of contents

1 Introduction

Human has always been great explorers; physical impossibility cannot stop this thirst for discovery. Some places are not accessible or dangerous to human, like space, deep oceans, battlefields, or nuclear accident zones. Autonomous vehicles are increasingly used in these domains to discover unreachable places or unsafe environments. The rover Curiosity [1] from NASA is an excellent example, it wasn't possible to send human on Mars at the time of launch and the delay of communication where to big for remote piloting. Therefore, this robot needed to autonomously explore a planet. It was still helped by the team on earth and wasn't fully autonomous, but this is probably one of the most advance autonomous vehicles ever made.

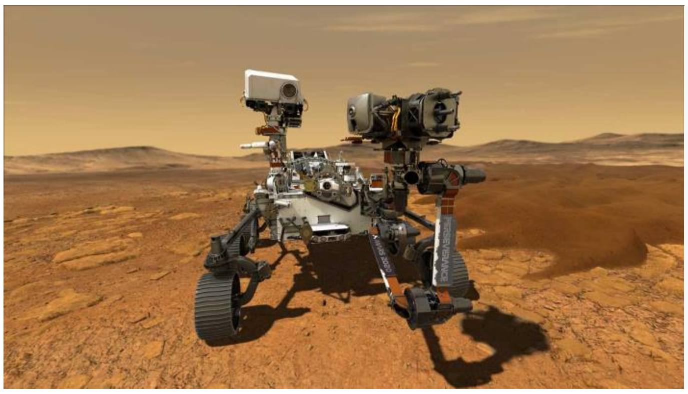


Figure 1 Curiosity rover, Nasa

Figure 2 Remus 600, KONGSBERG

On a different field the AUV (Autonomous Underwater Vehicle) REMUS 600 [2] is another good example. It can go down to 1500 meter deep and stay over 20 hours underwater without any human interaction. The type of exploration and the pick-up point are decided before the run. Then the robot is fully autonomous, able to take a payload or to create a map of the undersea ground. There exists various autonomous vehicle for various applications, in this master thesis the method of exploring any places will be more important than the vehicle exploring it.

Nonetheless, a prototype will be produced to test the algorithms. The case study will be an indoor 2-dimentional area, like a room for example. The robot will be a 2-wheel differential driving [3] robot only able to move on a plane surface. The robot equipped with a Lidar will be able to get a recording of the environment's distances and a 9-axis inertial sensor will provide the location. All the data will transit by a processing unit to a cloud computed environment to execute the machine learning. The equipment will be autonomously powered by a battery.

The robot's mission is to efficiently discover a room, without any human intervention. Here the goal will be to teach the robot to learn by itself. This technique is called reinforcement learning, it could be compared to the training of a dog that is rewarded for a good action and punished for a bad one. To do this, a reward function is programmed stating for example: "If the room is fully discovered in less than 10 minutes, 1 is given, if a crash in a wall occur, 0 is given". Then the robot will try to move around without any predefined directions. At the end of each iteration, it will be evaluated whether what was tried was worth a reward or a punishment. The behaviour realised during the reward function's iteration will be emphasised in the next step and the punished behaviour stopped. After many iterations the robot will learn strategies to adopt and move in a smarter way.

A second type of machine learning will be used in this project, clustering. This is a variety of algorithm seeking a small group of similar elements to classify them into different categories. The most famous technique is the K-Nearest Neighbors (KNN) algorithm. This algorithm returns the label of a point part of group with the smallest distance with each of its neighbour in this group. This part will explain how to create the map of an environment. A cloud of points will be outputted for the environment shape and another for the trajectory of the robot. It's necessary to transform those points into segments to create an actual map the robot can understand and navigate. To do so the algorithm will analyse all the points and try to find shapes or recuring pattern to associate it to an already known entity like a wall or an object.

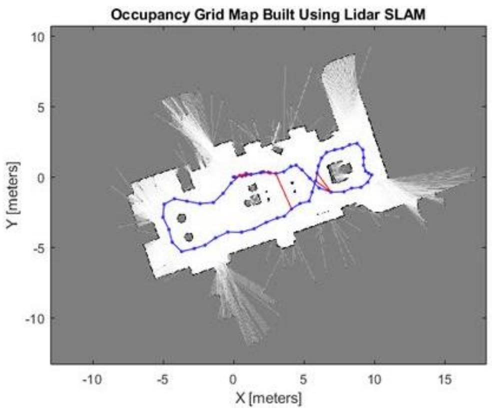


Figure 3 2D map build using Lidar SLAM

This master thesis aims to explore the domain of autonomous vehicle in machine learning. Two categories of machine learning will be explored, the reinforcement learning and clustering. The reinforcement learning goal is to make totally autonomous the training and the navigation of the robot. The clustering detects automatically different groups, classifying them as different element.

Some simulation could be done, to deepen the machine learning part. But to have an actual robot is better to learn about navigation techniques, sensors, the signal processing, IoT devices, real time communication and all the boundaries set by real hardware. The process will be, as human-free as possible, to dig the limit of creating an autonomous vehicle. The robot will not compute the model in the physical processing unit. This is a time and resource consuming activity that will be done remotely on an AWS (Amazon Web Services) server. To ease the communication process all the sensors will be connected as IoT (Internet of Things) devices to the AWS server in order to receive data in real time. The final goal is to return to the user a visual representation of the mapped area.

The following report will be presented as follow:

Part 1: An introduction to the topic of autonomous vehicle, with the aims and objectives of this project.

Part 2: A presentation of the related work and an overview of the existing technologies.

Part 3: The theorical preliminaries needed to understand the technical specificities of this report.

Part 4: Creation and explanation of the prototype with a focus on the sensors.

Part 5: Transformation of simple sensors into IoT devices communicating in real time.

Part 6: The process of creating the map of an environment.

Part 7: Experimental set up and how is tested the module.

Part 8: Discussion of the experimental results.

Part 9: Creation and training of the machine learning models.

Part 10: Conclusion of the project.

2 Related work

The SLAM (Self Localisation And Mapping) domain contains numerous studies as it is actively researched for 30 years now. Many techniques can be used combining different type of sensors, this is called sensor fusion. Odometry [4] is one of the fields explored using rotation of wheels to know the position of the vehicle. For aerial, submarine, or road purposes the GPS (Global Positioning System) is very good but not adapted to this study due to the small scale. The inertial sensor sensors are reliable on small scale and create some good results according to [5] for trajectory application.

For the detection of the environment the research show that static ultrasonic or laser beam are not adapted to this environment. In the past decade, the preferred solution is the Lidar as shown in [6]. It provides good details with a cloud of points able to precisely map an environment. Nevertheless, the future environment detection has already begun, and it will be with camera sensors. With some stereo camera able to measure depth in a picture some very interesting work is conducted on how to create vision-based 3D map [4].

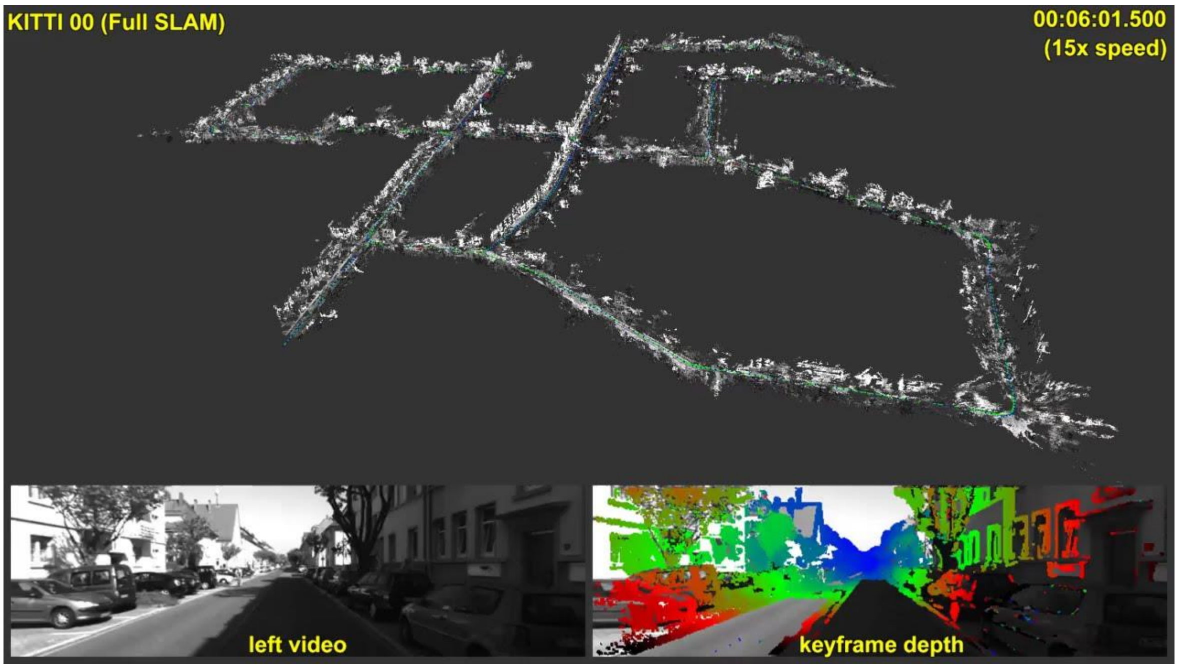


Figure 4 Visual SLAM using stereo cameras [37]

2D indoor SLAM is a very good theoretical basis explored in literature. Whether it is to compare the accuracy of the model to the real environment like in [7]. Or to build a reliable system over time as inertial sensor sensing tend to be inaccurate over long periods [5]. The different types of filtering for this SLAM can be easily compared in a 2D environment, whether it is Kalman filter [8], extended Kalman filter [9] or particle filter [10], only to mention the most famous ones.

The cloud computed approach is not very develop in scientific literature, but can is described in [11]. Some interesting IoT project are put together and develop using AWS or virtualisation services alike. The DeepRacer [12] platform is an AWS virtualisation platform where various teams learn how to cloud compute machine learning and compete over an interactive autonomous car race.

The machine learning fields is also full of very interesting work, the study from A. Notsu et al [13] is very adapted to this topic as it explains how to build and optimize a reinforcement model online. This makes this lengthy process faster and stronger.

3 Preliminaries

3.1 SLAM

Simultaneous Localisation And Mapping (SLAM) is a domain of research of robotic, knowing a burst of discovery for 30 years. The applications are endless, if a moving point is in an environment wanted to be known, SLAM can be applied. Autonomous driving, indoor and outdoor exploration, aero spatial and many more domains can use such technologies. J. K. Makhubela et al. [14] express the need to first define the kind of input signal for the type of SLAM. "*They are various sensors that are employed in a Simultaneous Localization and Mapping (SLAM) which characterized either as a laser, sonar and vision sensor.*"

3.1.1 *Environment*

In this project the choice is a 2-dimentional Lidar as it provides a relatively cheap sensor able to quickly get the shape of a 2-dimentonal room. This Lidar is a complex technology and become more complex as the resolution gets higher or the mapped environment becomes 3-dimentional. To explain simply, a laser measuring the distance from an object is put on an axis making 360° rotation. This creates a cloud of points expressing the environment, Y. Li et al. [6] describe the two main goals of this technology as *"1) real-time environment perception and processing for scene understanding and object detection and 2) high-definition maps and urban models' generation and construction for reliable localization and referencing."*.

3.1.2 *Position*

Then the position of the robot needs to be known. The detail of why the chosen sensor was selected will be explain later, all that is needed to say here is that the chosen sensor is a 9-axis inertial sensor. The addition of an accelerometer, a gyroscope, and a magnetometer. The accelerometer and the gyroscope define the trajectory of the robot while the magnetometer always gives the orientation of the robot. This last part eases the SLAM technique, the orientation of the input Lidar is crucial to calculate transformation between poses.
Creating a SLAM technique could be very interesting but would probably take all the time of a master thesis. Here the goal will be to find a model close enough from what is needed, modify it to the needs of the project and obtain some descent results.
The chosen model was created in the master thesis of J. Xu [7]. The model is recent (2019) and well explain in his master thesis to be easily reproduce. The Lidar and inertial sensor where different so some code modification where needed to make it work. The sensor fusion was based on extended Kalman filter. The model was created to compare accuracy between the ground truth, an ICP (Iterative Closest Point) and a feature-based line segment. The results are plotted and compared with a root mean square error. In this study case, the ICP algorithm was chosen as it is a very versatile and easy to muse model.
The extended Kalman filter (EKF) is one of the most efficient and used way to linearise sensor fusion for SLAM method [9]. This kind of filtering can be considered as a "*recursive estimator*" [21], this is very efficient when the result is expected. For a trajectory this ideal as it can be almost predicted where the robot will be in the next pose. This allows a quality signal processing and sensor fusion of the two sensors.
The ICP algorithm "*estimates the relative transformation between two overlapping point clouds*" [22]. The goal is to minimise the Euclidian distance between point to consider them as nearest neighbour. This method is computation intensive, fortunately the computing is done on powerful cloud-based servers.

3.2 Machine Learning

There is three main ways to train a machine learning model: supervised learning, reinforcement learning, and unsupervised learning. Supervised learning is composed of two phases: learning from labelled data set and inferencing to evaluate the accuracy of the model. Reinforcement learning is a method based on trial and error. The model is graded on each iteration and is rewarded for desired output. Unsupervised learning techniques are mainly grouping algorithm. Clustering is one of them, the algorithm is categorising data with similarities in groups. In this part will be explained the two-method used in this project, reinforcement learning for the navigation and clustering for the mapping.

3.2.1 *Reinforcement Learning*

The process can be compared to the psychological process used on animals or children. A reward is given for a desired result and a punishment for undesired result. At the beginning the model is ignorant and outputs some random results but from failure to success it learns what is expected. The goal is to make the model recognize by itself what is the expected result. This technique is prioritised when the environment and the state are constantly changing and is very useful in autonomous vehicle. For example, AWS DeepRacer is a platform to train racing car on a circuit [12][17].

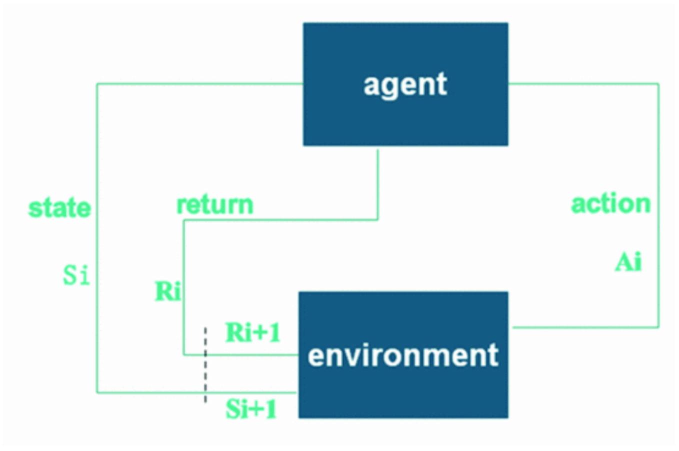


Figure 5 Markov process model from [2]

At the core of reinforcement learning method, the Markov Decision Process (MDP) [15] is present. This process of decision making is based on discrete time stochastic control [16], characterised by the following expressions:

$$R : S \times A \rightarrow S'$$

Where R the reward function depends on set of state S and A the set of action. Finally, S' is the state transfer function.

$$T : S \times A \rightarrow \Pi(S)$$

Then T map actions to probabilities, T (S, A, S') the probability of arriving to state S' by doing action A in state S.

3.2.2 *Clustering*

Clustering is a method of categorising similar data into different groups. Huge sets of data can be treated by this method as no labelling is required. The clustering method evaluate the distance between each element, then estimates a centroid for this cluster. The output of a clustering algorithm is the statistical description of the cluster centroids and how many components are in each cluster.

Different methods exist [19]:

- Hierarchical clustering is giving as a result a tree expressing the similarities at different levels between the points in the clusters.
- Partitional clustering is when the user desires a specific number of output clusters for example the k-means clustering [20].
- Fuzzy clustering is called a soft clustering as opposed to the two previous considered hard clustering. Here one value can be in different clusters.

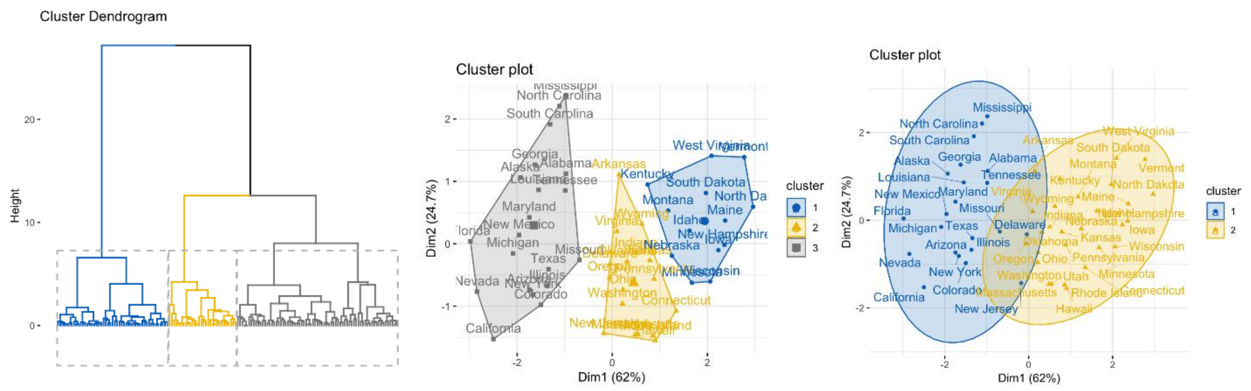


Figure 6 From left to right: Hierarchical clustering, Partitional clustering, Fuzzy clustering

The most used clustering algorithm is the KNN using K-means method [18] to calculate the clusters. Here the goal is to classify unlabelled data in a defined number of clusters. In the case of the study the goal is to create a map therefore in the cloud of points will be isolated the points forming walls. This process is iterative and works the following way:

- Chose k random points for the average of each partition $m_1, m_2, \ldots, m_k$.
- For each iteration each point is distributed in a cluster until convergence.

$$S_i^{(t)} = \{x_j : \left\| x_j - m_i^{(t)} \right\| \leq \left\| x_j - m_{i^*}^{(t)} \right\| \forall i^* = 1, \ldots, k\}$$

- For iteration update the new centroid of the cluster.

$$m_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j$$

With x_i being a specific point in the sample x, contained in $S_i^{(t)}$ the set of all the closest partition.

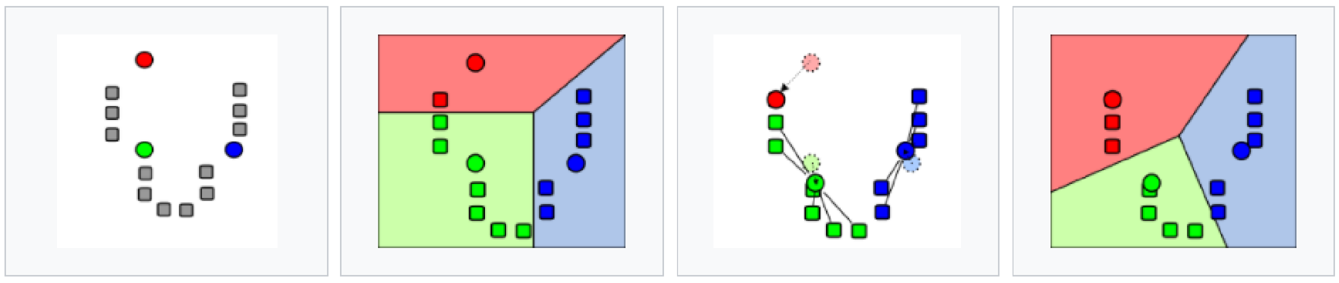


Figure 7 K-mean clustering process

4 Prototyping

To measure real values for the machine learning model a prototype was needed. It could have been simulated with many platforms offering an alternative. For example, the video game Trackmania [23][35] offers an interface to implement machine learning model. DeepRacer [12] is a service of AWS created to teach reinforcement learning, by training a car to evolve by itself on a circuit. Nevertheless, the creation of a prototype is always a fascinating experience worth the few difficulties it brings.
This prototype was created with SLAM technics in mind. Therefore, it needed to know its position thanks to an inertial sensor as well as the position of the surrounding with a 2-dimensional Lidar. This is a mobile robot, so a mobile base is provided by the educative robot Thymio II. To make all these parts communicate with each other a processing unit is required, taking the form of a Raspberry Pi 4. This robot is an autonomous vehicle, so it needs to be autonomous in energy with a power bank. All of this is wired by various cables and connected by WIFI to internet.

4.1 Sensors

4.1.1 *Inertial sensor*

The choice of the position sensor evolved with the prototype. During the first semester in the Autonomous Vehicle module a prototype was created using odometry principle with wheel encoder. This technic measures the rotation of the wheel to determine the position of the robot. This process was found to be not precise enough for the project as the result is very dependant on the ground and wheel texture as the robot might slip creating some inaccuracy.
In the first phase of the master thesis a 6 axis IMU (Inertial Measurement Unit) sensor (MPU6050[24]) was chosen. This is the combination of a 3-axis accelerometer and a 3-axis gyroscope. The accelerometer measures the acceleration in of the robot in three dimensions. Here the robot is moving only in two dimensions, but the Z axis is still measuring the gravitational attraction and need to be consider for an accurate trajectory calculation.

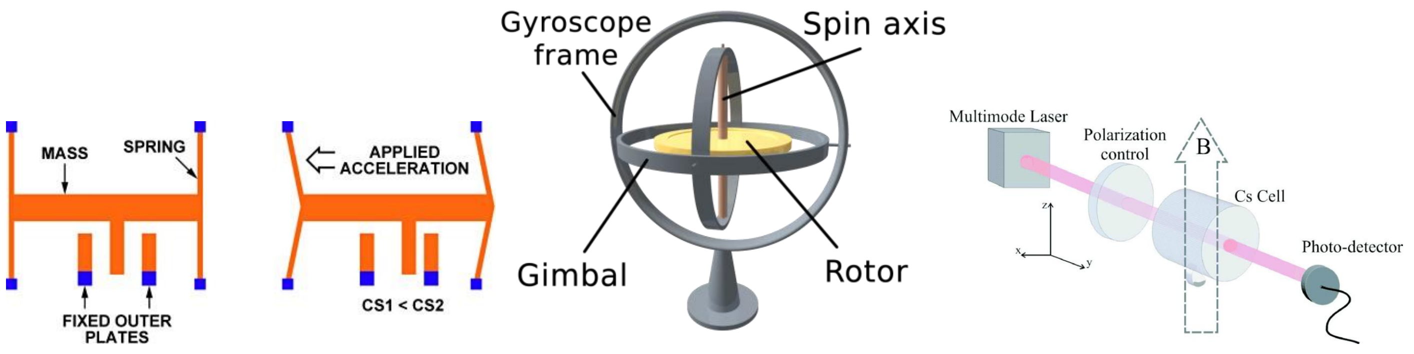


Figure 8 Scheme of Accelerometer Gyroscope Magnetometer

The gyroscope records the roll, pitch, and yaw, which are the speed of rotations around the X, Y, and Z axis. This was used for a time, as the research in SLAM started to move forward the orientation of the robot between each pose [25] was lacking and the gyroscope was not sufficient.

Therefore, the last choice was a 9-axis IMU sensor (MPU9250 figure 9 [26]). The difference with the previous one is the addition of a 3-axis magnetometer. This solution was found by the study and adaptation of the SLAM method of [7]. The magnetometer is an electronic compass allowing to always know where the cardinal points are. To calculate the transformation matrices between the poses, the orientation is key [25]. This allowed to be truly autonomous as the robot doesn't need a starting point and knows the trajectory defined by the combination of the gyroscope and the accelerometer. The robot's orientation is known by the magnetometer.

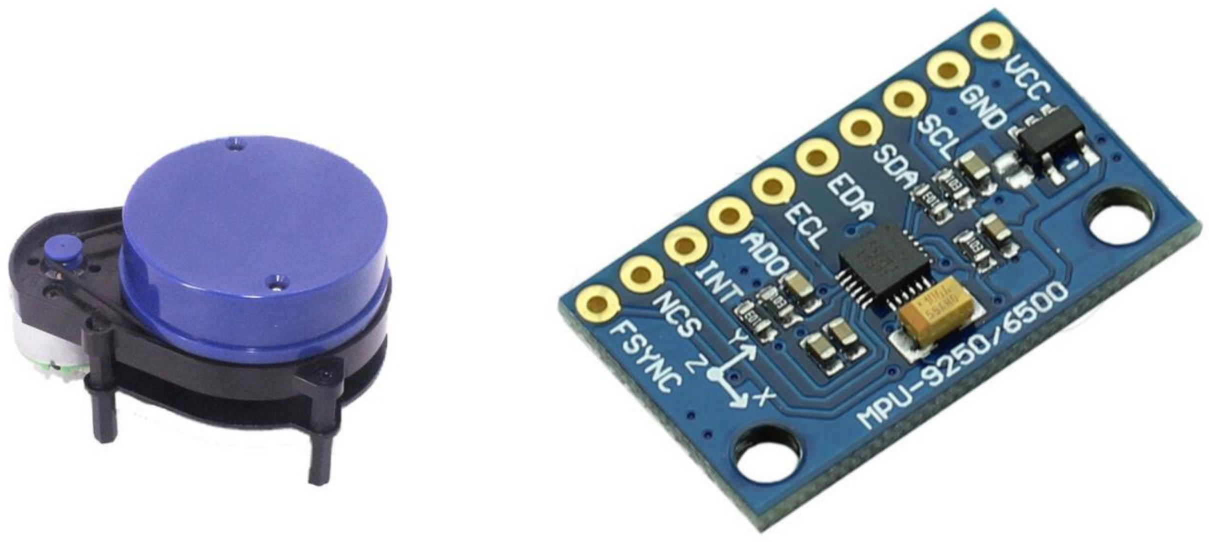


Figure 9 YD Lidar X4 and MPU 9250

4.1.2 *Lidar*

In the first version of the robot created in the autonomous vehicle module, the sensor to record the surrounding was first a time-of-flight ultrasonic sensor [27]. Which is cheap and simple to use, but the recordings are simply inaccurate. Then was used a time-of-flight laser mounted on a servo motor. This was an interesting solution to build a laser recorder for under 10 pounds, but the results were also inaccurate. Therefore, for the master thesis the needed recording to be good enough and the choice was a 2-dimensional Lidar. Over the last decades the price of Lidar considerably dropped and are now accessible to hobbyists.

Lidar means LIght Detection and Ranging, the principle is making a 360° rotating laser beam recording the dimensions of a room. The laser records the range distance of the 2-D environment in a cloud of points.

$$X_i = D_i * \mathrm{i}\cos\theta * \cos(a)$$
$$Y_i = D_i * \mathrm{i}\sin\theta * \cos(a)$$

The position of the robot is expressed by the Cartesian coordinate X_i and Y_i. Then the distance is from the object is return in D_i as well as the angle θ. The raw output of the Lidar is shown in figure 10.

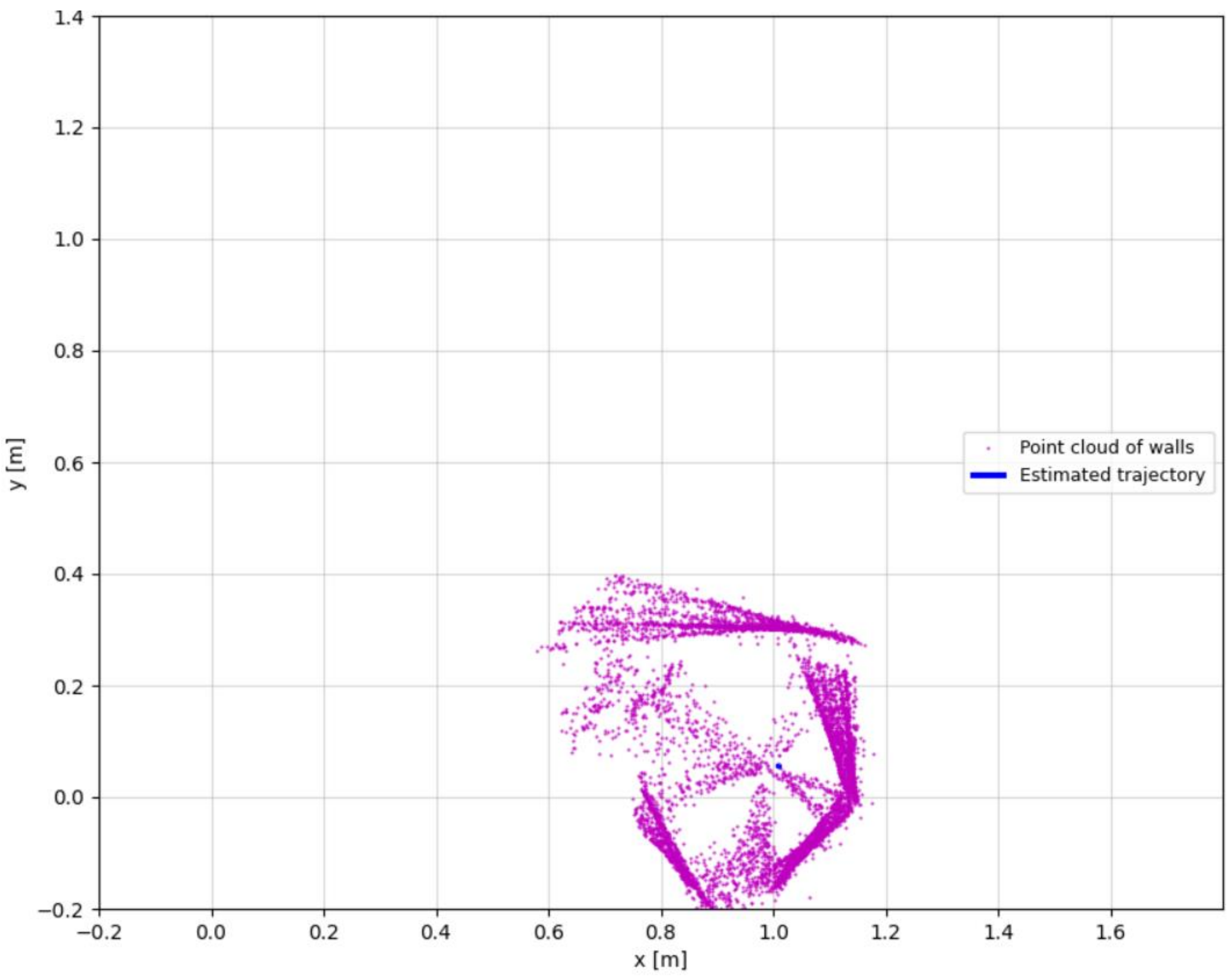


Figure 10 Multiple clouds of points of Lidar without processing

The most appropriate choice for this project was the YD Lidar X4 shown in figure 9 [36]. The range of the Lidar measuring is from 12 cm to 10 m. A full 360° recording takes 2.7 seconds and in the most common area of use between 0.5m and 6m the relative error is 1.5% of the distance. The angle resolution is also good enough with 0.5°. Overall, this provides an interesting solution under 90 pounds to find accurately the surrounding. The result in real time is tricky as the robot needs to be still for 2.7 seconds. This will be a problem in the rest of project, particularly in addition with the cloud computing also adding delay in the communications.

4.2 Mobile robot

The mobility of the robot is insured by an educative platform called Thymio II shown in figure 11 [28]. This robot was created by EPFL (Ecole Polytechnique Federal de Lausanne) and is using 2 wheels differential driving to move in 2-dimensional plane. This robot was used during all the phases of the development as it is easy to develop and very versatile. The code was enhanced in each phase to make it the most efficient. The robot is packed with some different sensors but none of them were used in this project, so this robot is just used for the wheels and its good carrying capacity.

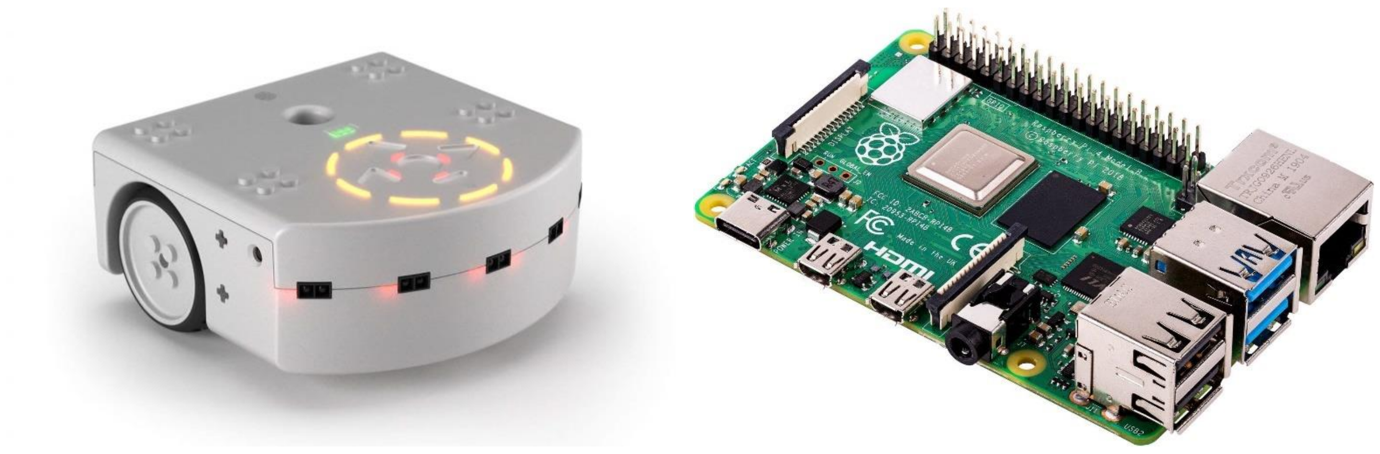


Figure 11 Thymio II (left) and Raspberry Pi 4 (right)

4.3 Processing unit

The Raspberry Pi 4 shown in figure 11 [29] was chosen as the processing unit for the prototype. This is a small computer, the size of a credit card but offer some very interesting features. It has many different ports to be plugged to different sensor previously mentioned. It also has a Wi-Fi module to be connected to internet and a good 8Gb RAM if some local processing was required. This is a very popular project base for hobbyist and therefore has a lot of resources to create any kind of projects.

4.4 Power Supply Wiring Assembly

This robot mut be autonomous in its environment so a storage unit is mandatory. The chosen power bank was a 10 000mAh and 18W Xiaomi, with two ports to power two elements at once, the Lidar and the Raspberry PI. The Thymio II and the inertial sensor require not a lot of power supply and are powered by the Raspberry Pi.

The wiring can be observed in the figure 12. The data from the inertial sensor are communicated through I2C protocol using jumper cables, while the data of the Lidar are communicated through USB-C port. The alimentation is via a USB-C cable for the Raspberry Pi and Micro-USB for the Lidar, the communication with the Thymio is also done with Micro-USB.

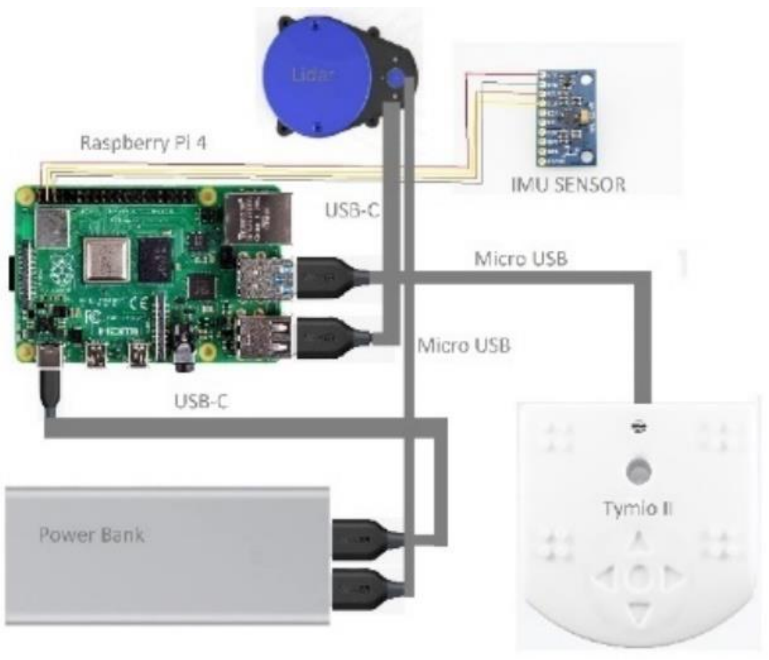


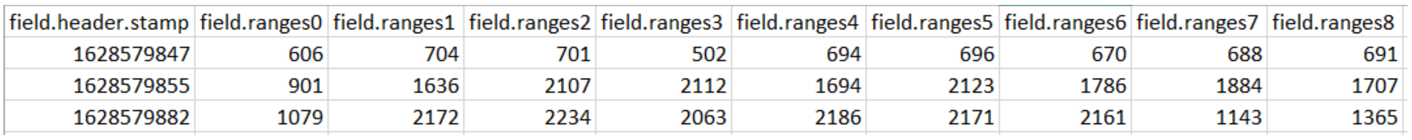
Figure 12 Wiring scheme of the prototype

5 IoT Sensors

Most of the time in machine learning the computing power required to execute a task can be too important for a small processing unit. That's why all the data of the robot are cloud computed rather than computed locally by the Raspberry Pi. The goal here is to make all the data converge in real time on the server with the computing power needed for training a machine learning model.

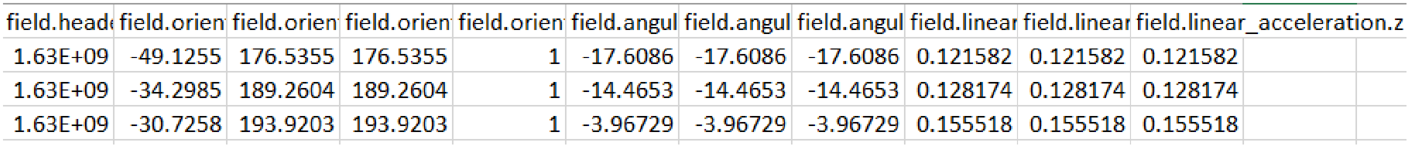
5.1 Sensors

The Lidar uses the PyLidar3 library [39] to write a Python script lunching the recording, store it in a Pandas Dictionary [40] and send a MQTT (Message Queuing Telemetry Transport[41]) request to the IoT Core service [42]. When the request is accepted, the data in the JSON [43] format is sent using the MQTT protocol. The same is done for the inertial sensor on a different script so that both data are obtained at the same time.
First, for the Lidar the first column contains the time stamps. Then 360 distances associated to the angle it was taken from. This measurement is not instant it takes 2.7 seconds and requires the robot to be still during this moment. Second, the inertial sensor is sending a timestamp followed by the 3 values of the magnetometer, 3 values of the gyroscope, 3 values of the accelerometer. These values are instant and need to be associated correctly with the Lidar values to get a coherent result.



field.header.stamp	field.ranges0	field.ranges1	field.ranges2	field.ranges3	field.ranges4	field.ranges5	field.ranges6	field.ranges7	field.ranges8
1628579847	606	704	701	502	694	696	670	688	691
1628579855	901	1636	2107	2112	1694	2123	1786	1884	1707
1628579882	1079	2172	2234	2063	2186	2171	2161	1143	1365

Figure 13 Lidar data output [timestamp, angle from 0 to 359 : range distance]



field.head	field.orien	field.orien	field.orien	field.orien	field.angul	field.angul	field.angul	field.linear	field.linear	field.linear_acceleration.z
1.63E+09	-49.1255	176.5355	176.5355	1	-17.6086	-17.6086	-17.6086	0.121582	0.121582	0.121582
1.63E+09	-34.2985	189.2604	189.2604	1	-14.4653	-14.4653	-14.4653	0.128174	0.128174	0.128174
1.63E+09	-30.7258	193.9203	193.9203	1	-3.96729	-3.96729	-3.96729	0.155518	0.155518	0.155518

Figure 14 Inertial sensor data output [Magnetometer XYZW, Gyroscope XYZ, Accelerometer XYZ]

5.2 AWS Communication

To do this the module of IoT Devices by Mrs Chathurika Goonawardane was used to create the IoT communication of the project. The scheme in the figure 15 represents how the data are transferred by AWS from the physical sensor to end up on the Sagemaker studio where the machine learning can be trained.

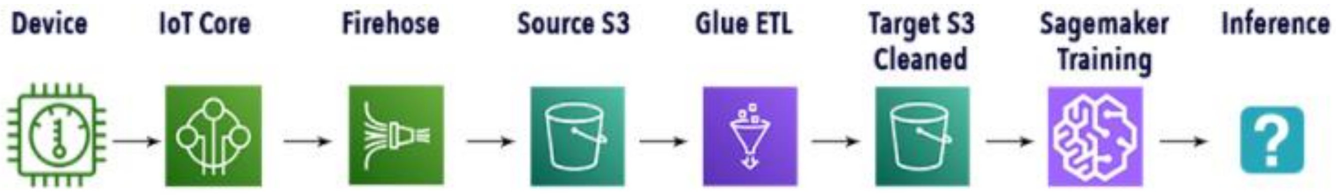
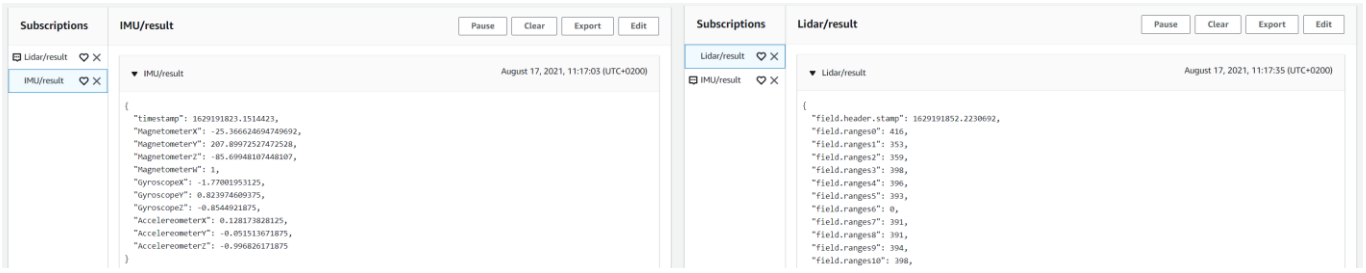


Figure 15 Cloud process from sensor to storing values in real time

First using IoT Core and a Python script data are sent from the sensor through WI-FI to AWS. Then Kinesis Firehose is sending the data to be stored in a data lake on the S3 server. Then the Glue ETL is used to transform the data into JSON to have clean data for the training then data are stored in S3 bucket and can be accessed by the Jupiter Notebook created using Sagemaker.

5.2.1 *IoT Core*

The data from both sensors are obtained from a Python script on the Raspberry Pi. The same code (see repository [34]) is sending data to the IoT Core. First a "Thing" is created, to ensure the security of the data with a private key and a certificate. This ensures that the data are privately transferred to the AWS account endpoint. The values of the sensor are transformed by the JSON dump process [43] to be published using the Wi-Fi to the desired topic via the endpoint of the AWS account. The output data of both sensors can be observed in real time using in the MQTT test client as seen in figure 16.



```
{
  "timestamp": 1629191823.1514423,
  "MagnetometerX": -25.366624694749692,
  "MagnetometerY": 207.89972527472528,
  "MagnetometerZ": -85.69948107448107,
  "MagnetometerW": 1,
  "GyroscopeX": -1.77001953125,
  "GyroscopeY": 0.823974609375,
  "GyroscopeZ": -0.8544921875,
  "AccelereometerX": 0.128173828125,
  "AccelereometerY": -0.051513671875,
  "AccelereometerZ": -0.996826171875
}
```

```
{
  "field.header.stamp": 1629191852.2230692,
  "field.ranges0": 416,
  "field.ranges1": 353,
  "field.ranges2": 359,
  "field.ranges3": 398,
  "field.ranges4": 396,
  "field.ranges5": 393,
  "field.ranges6": 0,
  "field.ranges7": 391,
  "field.ranges8": 391,
  "field.ranges9": 394,
  "field.ranges10": 398,
```

Figure 16 Output data of Lidar and IMU in the MQTT test client

5.2.2 *Kinesis Firehose*

Kinesis Firehose ensures the delivery stream of the data. Here the goal is to create a data lake with accurate transmission to avoid gradual misalignment or concept drift [20]. Basically, the data need to match the sending time to avoid confusion with other data. This pipe is created to detect when data is sent to the IoT Core and immediately label and transfer it to the desired S3 Bucket. This part can monitor the volume of data delivered to server by plotting: bytes/second, requests/second, records/second in figure 17.

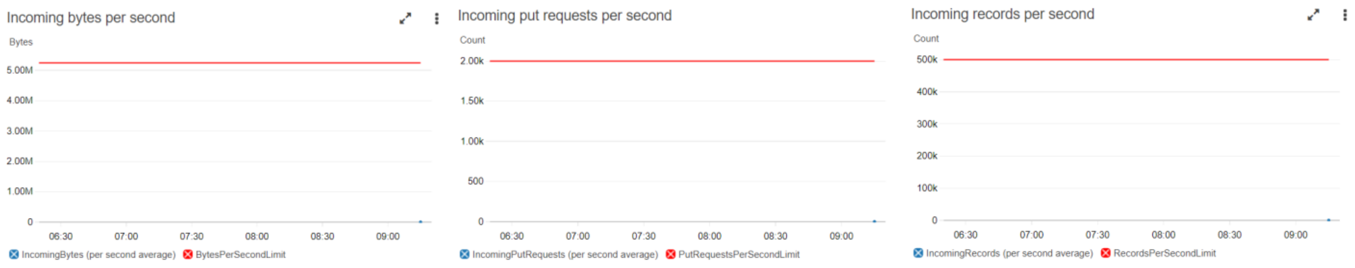
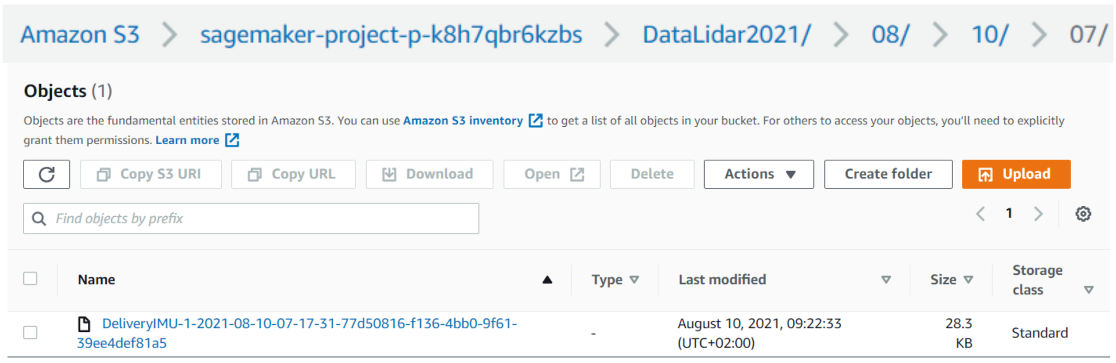


Figure 17 Kinesis Firehose delivery stream monitoring

5.2.3 *Source S3 Bucket*

In this part a data lake is created, this is one of the fundamental of any big data project. A dedicated place needs to be accessible by different services (IoT Core, Glue, Sagemaker...) to store a lot of data. This data storage can store any format, in the case of the project CSV, JSON or PARQUET [44]. The data are stored in separated folder named after the time of recording like in figure 18.



Amazon S3 > sagemaker-project-p-k8h7qbr6kzbs > DataLidar2021/ > 08/ > 10/ > 07/

Objects (1)

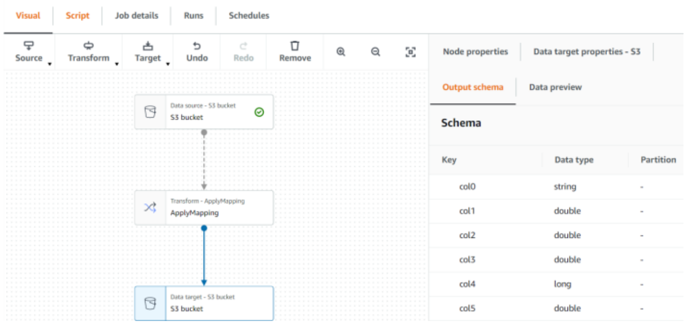
Objects are the fundamental entities stored in Amazon S3. You can use **Amazon S3 inventory** to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. **Learn more**

	Name	Type	Last modified	Size	Storage class
☐	DeliveryIMU-1-2021-08-10-07-17-31-77d50816-f136-4bb0-9f61-39ee4def81a5	-	August 10, 2021, 09:22:33 (UTC+02:00)	28.3 KB	Standard

Figure 18 Path to direct access raw data in S3

5.2.4 *Glue ETL*

Dealing with a lot of raw data is never easy, therefore some pre-processing is required. The ETL process (Extract, Transform, Load) from AWS Glue [45] can send pre-processed data in the right format from a raw data lake to a desired storing area. First, the Glue Crawler read data from the source bucket of the data lake to create a Glue Schema. Then the data are transformed from JSON to CSV or PARQUET and mapped from the schema to the new database. The PARQUET format is preferred as it is best adapted to handle big data. The queries are faster. Complex data structures are available. Aggregation is less time consuming. Overall, this format is made to be efficiently used by machine learning model. The transformation can be created by the Glue studio either by coding or using a visual scheme.



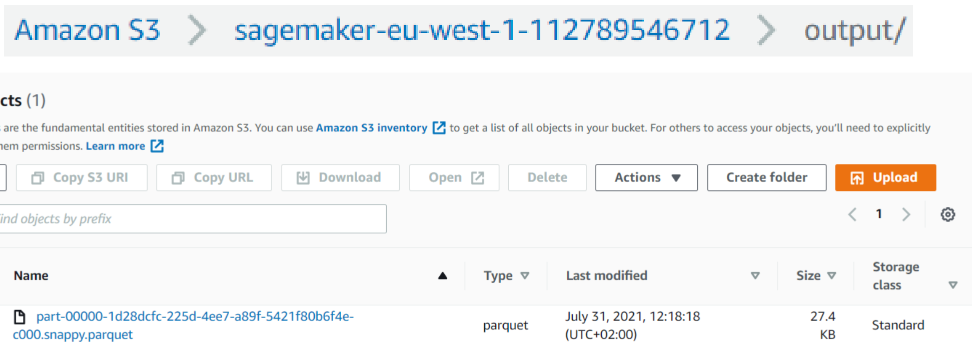
Schema

Key	Data type	Partition
col0	string	-
col1	double	-
col2	double	-
col3	double	-
col4	long	-
col5	double	-

Figure 19 Glue studio visual formatting

5.2.5 *Output S3 Bucket*

In this bucket the data are cleaned, labelled and in the correct format. They are ready to use by the training model.



Objects (1)

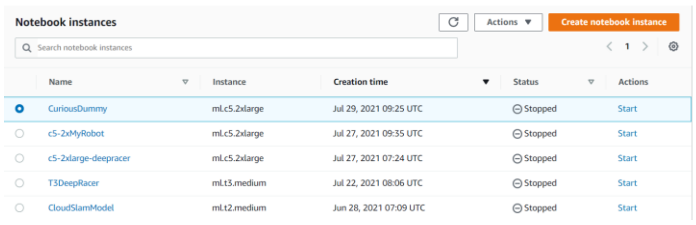
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
part-00000-1d28dcfc-225d-4ee7-a89f-5421f80b6f4e-c000.snappy.parquet	parquet	July 31, 2021, 12:18:18 (UTC+02:00)	27.4 KB	Standard

Figure 20 Output values in a different format and different bucket ready for the machine learning

5.2.6 *Sagemaker Studio*

This is an interface where the data from any bucket can be accessed by a simple Python query. This is the place were all the online Python code will be run. This will be essential for signal processing, SLAM, map creation and reinforcement learning training. Different size of instances is available for the required computing power, different instances were tested. The chosen instance was a ml.c5.2xlarge which is an optimized calculus instance with 4 vCPU ang 16GiB of RAM.



	Name	Instance	Creation time	Status	Actions
●	CuriousDummy	ml.c5.2xlarge	Jul 29, 2021 09:25 UTC	Stopped	Start
○	c5-2xMyRobot	ml.c5.2xlarge	Jul 27, 2021 09:35 UTC	Stopped	Start
○	c5-2xlarge-deepracer	ml.c5.2xlarge	Jul 27, 2021 07:24 UTC	Stopped	Start
○	T3DeepRacer	ml.t3.medium	Jul 22, 2021 08:06 UTC	Stopped	Start
○	CloudSlamModel	ml.t2.medium	Jun 28, 2021 07:09 UTC	Stopped	Start

Figure 21 Different notebook instances

6 Map Creation

A map is the accurate representation of a spatial area with respect to a certain reference frame. In autonomous vehicle the domain is referred as SLAM it is the type of algorithm needed to estimate its position and the distance of the surrounding in order to create autonomously a map. In a first time the algorithm needs to operate the fusion of the two sensors and optimize the model by using an extended Kalman filter (EKF). This process considers the error and uncertainties of the two sensors and output the most probable estimation of the state. This help minimizes the errors (loss function) always present in physical recording.

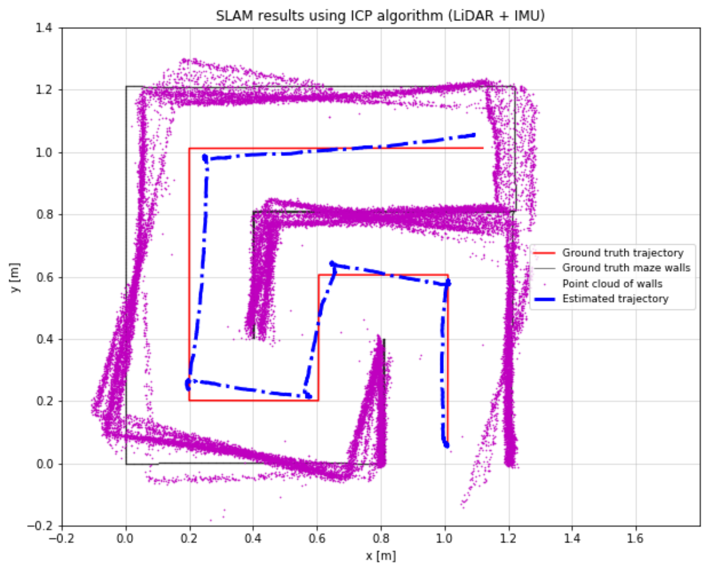


Figure 22 SLAM Map output example from [7]

The second part of SLAM is the Lidar scan matching algorithm. In this study, Iterative Closest Point (ICP) will be introduced, this algorithm creates a link between the cloud of points to form solid shapes. The goal of this paper is not to create a SLAM method so the work of J. Xu [7] will be adapted and used with his permission and help. This work is Master Thesis exclusively dedicated to creating a SLAM method and close enough to the topic to be reused. Nevertheless, the mathematical demonstration will be less pushed here and can be referred to in [7].

6.1 Extended Kalman Filter

6.1.1 *Kalman Filter*

This algorithm presented by Rudolf Kalman in 1960 [31] is a widely used in various engineering fields. In robot control and path planning, times series or computer vison this tool is a key element to dynamically predict the output based on multiple input sensors. The goal is to create a more accurate estimation than the recorded value of the sensors. This process is fast to compute and therefore can be used in real time application.

To estimate the true system state x_k, first is considered F_k the physical transition matrix applied to the previous step. Then is added the control input model B_k multiplied by the control input vector to verify the motion of the robot. Finally, w_k is added the process noise assumed to be a 0-means Gaussian distribution.

$$x_k = F_k x_{k-1} + B_k u_k + w_k$$

In a first time the state $\check{x}_k$ is predicted based on the previous estimation $\hat{x}_{k-1}$ with the following equation:

$$\check{x}_k = F_k\hat{x}_{k-1} + B_k u_k$$

Then this state is corrected using the optimal Kalman gain matrix K_k, using the predicted covariance $\breve{Pk}$, H_k is the observation model, and the covariance of the observation noise R_k.

$$K_k = \breve{\mathbf{P}}_k H_k^T (H_k \breve{P}_k H_K^T + R_k)^{-1}$$

This matrix is used to calculate the error state δx_k.

$$\delta x_k = K_k(y_k - H_k\check{x}_k)$$

Finally, it can be estimated the corrected state $\hat{x}_k$ and the corrected covariance P_k.

$$\hat{x}_k = \check{x}_k + \delta x_k$$
$$\widehat{P_k} = (I - K_k H_k)\check{P}_k$$

Mathematically this is the best unbiased estimator. But this only works in linear dynamic system, but this almost never exists in real life, most of the system are nonlinear. It can also be added that covariance is rarely precise and requires some iteration to be tunned. The Kalman filter need to be extended to be applied to nonlinear problem.

6.1.2 *Extension of the Kalman filter*

The following nonlinear system is composed of f and h respectively the motion model and the observation of the model function, they are both differentiable.

$$x_k = f(x_{k-1}, u_k) + w_k$$
$$y_k = h(x_k) + v_k$$

The goal here is to linearize approximatively these functions using first order Taylor series expansion. The nominal state $\hat{x}_{k-1}$ and the control input u_k are then used to transform the transition matrix F_k into a Jacobian matrix. The observation matrix H_k also becomes a Jacobian matrix therefore the model is linearised and can be computed.

6.2 ICP

Developed in 1992 by P. J. Besl & N. D. McKay in [32] Iterative Closest Point is nowadays one of the most classic and studied method to combine cloud of points. The algorithm is created to match the clouds of points of two different scans. The goal is to find the best rigid transformation to minimize the least square criterion between two same points in space. This method is perfect for Lidar and return the optimal transformation matrix (rotation + translation).

In a first time a search for correspondence is conducted: for each point in the set $P_k = \{p_{k,i}\}_{i=1}^{N_k}$, the algorithm will try to match it to the closest point in P_{k-1}. The index of the corresponding point is store in c(i). This is known as clustering, the most famous method being KNN (K-nearest neighbor) [32]. Then the rigid transformation estimation aims at finding the rotation matrix R_k^{k-1} and translation vector t_k^{k-1}minimizing the least square error.

$$\min_{R_k^{k-1}, t_k^{k-1}, \{c(i)\}_{i=1}^{N_{k-1}}} \sum_{i=1}^{N_k} \left\| \left(R_k^{k-1} p_{k,i} + t_k^{k-1} \right) - p_{k-1,c(i)} \right\|_2^2$$

Then the scan points are transform using the transformation array. The process is repeated until the solution converges. With this technique is obtained the rotation matrix and translation vector optimal for the transformation between two sets.

6.3 Mapping

Once all these principles are laid down the building of the map can start. The position of the robot can be estimated in a 2D reference frame ($x_k, y_k \in p_k$) in addition the yaw angle (ψ_k) expressing the orientation of the robot in this reference frame. Thanks to the ICP the transformation between the frames is also known.

The first step to create this navigation environment is to initialise the reference frame by choosing an origin. This point is decided to be the starting point of the first Lidar scan. Adding each cloud of points means adding rotation matrices and translation vectors. Consecutive rotation matrices can be simplified as the product of those rotation matrices. The goal is to express all the scan points $p_{k,i}^{[k]}$ with respects to the original scan point, that will be considered the global reference frame $p_{k,i}^{[0]}$. Therefore, the following equation is obtained to build the map.

$$p_{k,i}^{[0]} = \prod_{j=1}^{k} R_j^{j-1^T} p_{k,i}^{[k]} - \sum_{j=1}^{k} \left(\prod_{m=1}^{j} R_m^{m-1^T} \right) t_j^{j-1}$$

$$\forall p_{k,i}^{[k]} \in P_k$$

6.4 Used SLAM model

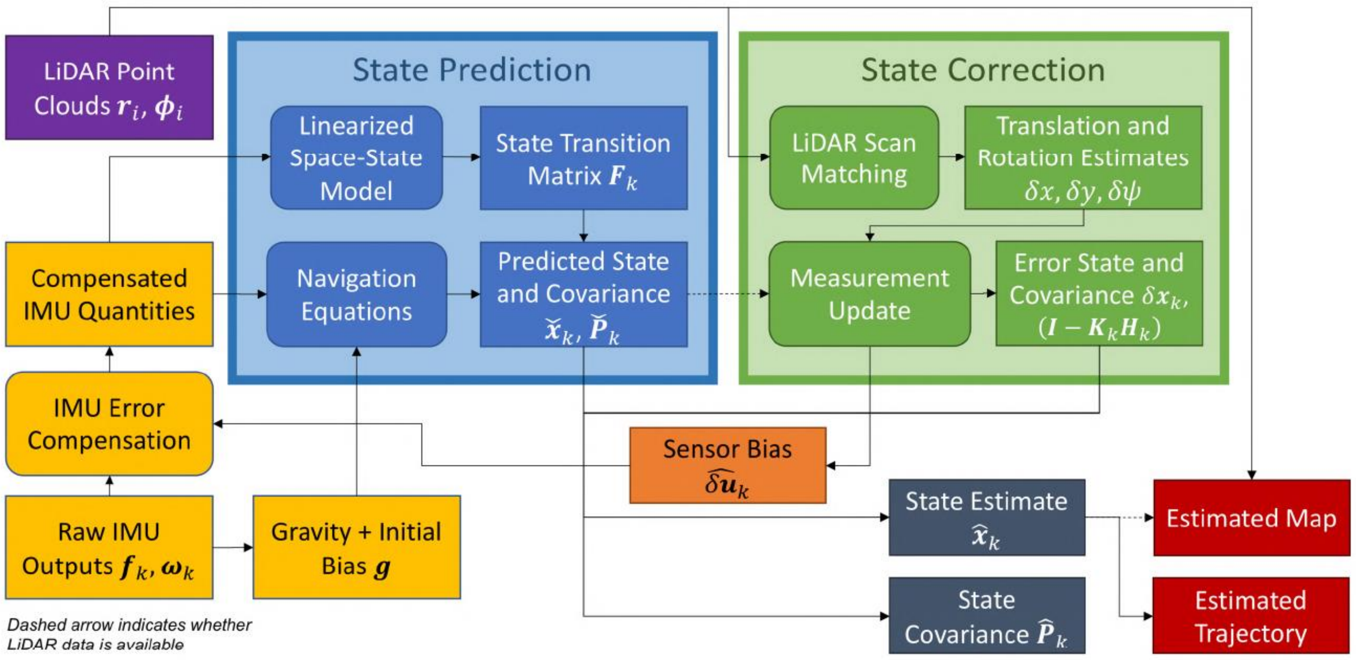

LiDAR Point Clouds r_i, ϕ_i
State Prediction
Linearized Space-State Model
State Transition Matrix F_k
Navigation Equations
Predicted State and Covariance $\check{x}_k, \check{P}_k$
State Correction
LiDAR Scan Matching
Translation and Rotation Estimates $\delta x, \delta y, \delta\psi$
Measurement Update
Error State and Covariance δx_k, $(I - K_k H_k)$
Compensated IMU Quantities
IMU Error Compensation
Raw IMU Outputs f_k, ω_k
Gravity + Initial Bias g
Sensor Bias $\widehat{\delta u}_k$
State Estimate $\hat{x}_k$
State Covariance $\hat{P}_k$
Estimated Map
Estimated Trajectory
Dashed arrow indicates whether LiDAR data is available

Figure 23 Diagram of system architecture of the SLAM model used in [7]. Boxes with sharp corners represent physical quantities, whereas boxes with rounded corners describe procedures.

The SLAM model presented in the figure 23 is the one created in [7]. It can be observed that the input values are the Lidar point of clouds and inertial sensor values. Then some pre-processing is done compensating inertial sensor error and adding gravity and initial bias. Those corrected values are used to predict the state of the robot (position + orientation), this is done using extended Kalman filter. Then the Lidar point of clouds are matched to there previous corresponding points using the ICP algorithm. The environment is updated at each step, updating at the same time the sensor bias. Then the state estimation and state covariance are computed for future estimation. Finally, the estimated trajectory and estimated map can be graphically expressed.

7 Experimental Setup

The goal of this part is to map a real-life environment, so an experiment had to be conducted. This is not a simulation, so the prototype needs to be faced with a physical environment and treat raw data. The robot will be launched to autonomously discover the area while harvesting multiple clouds of points of the environment as well as its trajectory. The data will be communicated in real time using the AWS services to create a map of the environment.

7.1 Robot

The choice of the robot components and its assembly were discussed previously in part 5. Here will be discussed how the robot is operated to move autonomously. A prototype was built relatively quickly thanks to the knowledge learnt during the two first semesters. The mistakes and the time-consuming experimentations realised in other module allowed to chose and assemble the hardware relatively quickly. Most of the element of the robot were already owned and the adaptation for this project was efficient. In two weeks, all the hardware of the robot was assembled, and the robot could harvest data and be remotely piloted.

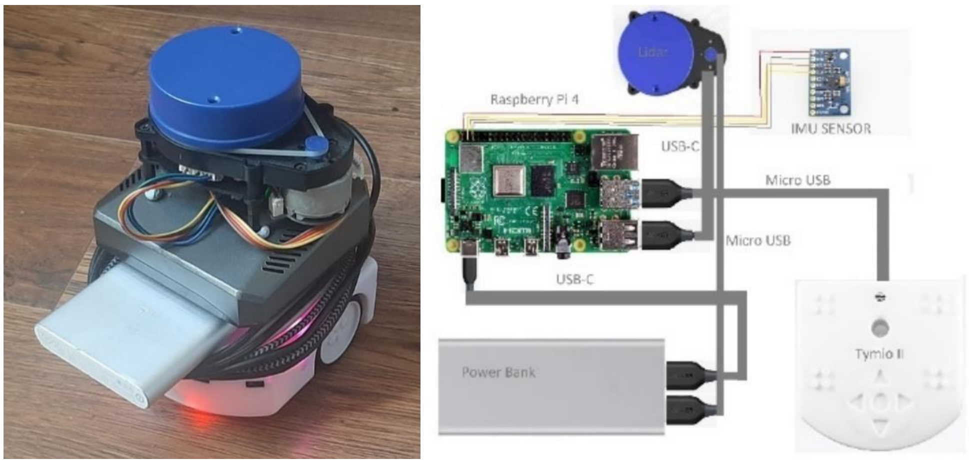


Figure 24 Prototype assembly: physical, scheme

The first time the robot was remotely piloted using a home-made virtual joystick on a locally hosted web page accessible to any device connected to the same Wi-Fi. Then the first step toward automation was to let the robot move randomly in the area. The final goal was to create the reinforcement learning model to let the robot learn how to move into space.

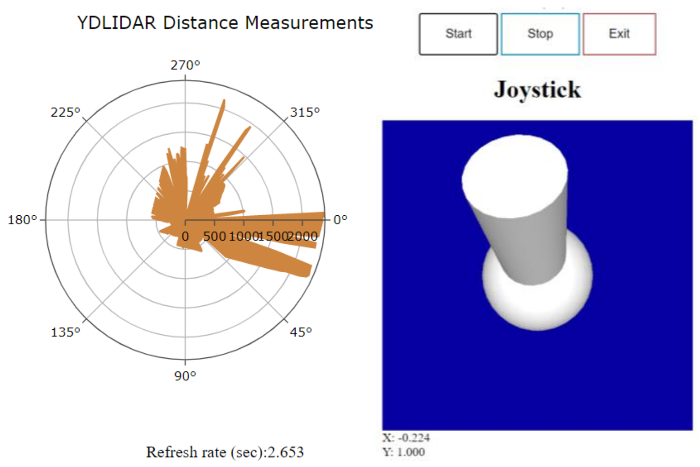


Figure 25 Local web page remotely controlling the robot

After one month of trying to make the machine learning work, some progress was made but two things became clear. The ambitions of the author were above his developing skills and the time was too short to realise such a project. The sum of knowledge in programming (Python 3), machine learning and data science were just too big, so the target was adjusted. The map creation became the priority to have a visual output with a physical prototype. To give up the machine learning part would have been such a waste after all the knowledge learnt. What was decided was to only describe the architecture and the intended functioning of the machine learning modelin the part 9. The development of a working solution is left for a further work and probably the future job of the author.
The final solution was to let the robot move in a random manner to let it harvest a lot of data and to stop the robot when it crashes into a wall. The robot is still remotely pilotable. This is useful when the pilot wants the robots to avoid some obstacle or to push the robot in an unexplored area.

7.2 Discovery area

The chosen environment was a bedroom, but some modifications were required to ease the data harvest. In a first time all the small elements under 15cm were removed as they wouldn't be seen by the LIDAR placed on top of the robot. Then it was discovered that the robot tended to disconnect (hardware connection problem) over a long period of time, so the choice was made to cut in half the dimension of the room to shorten the discovery time.

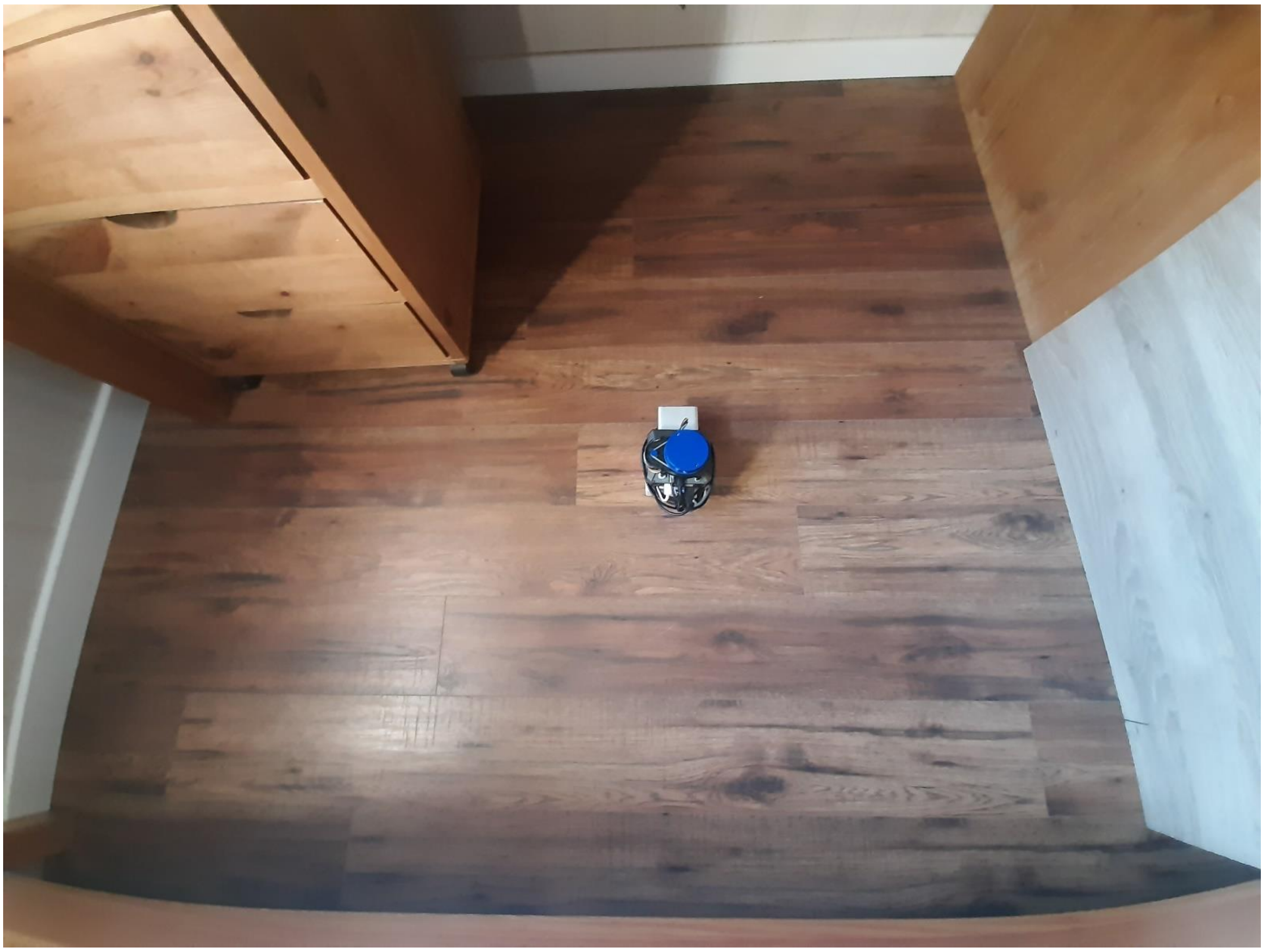


Figure 26 Experiment discovery area

7.3 Data Harvest

Once the robot is functional and the environment is adapted to the recording the data harvest can start. The creation of a cloud computed interface with all the part of the robot connected was programmed using AWS. This part also took two weeks to have real time communication and storage of the data on the various services of AWS. This part was eased by a previous 3-month internship on the subject and the "Internet of Thing Device" module conducted by Mrs Chathurika Goonawardane.

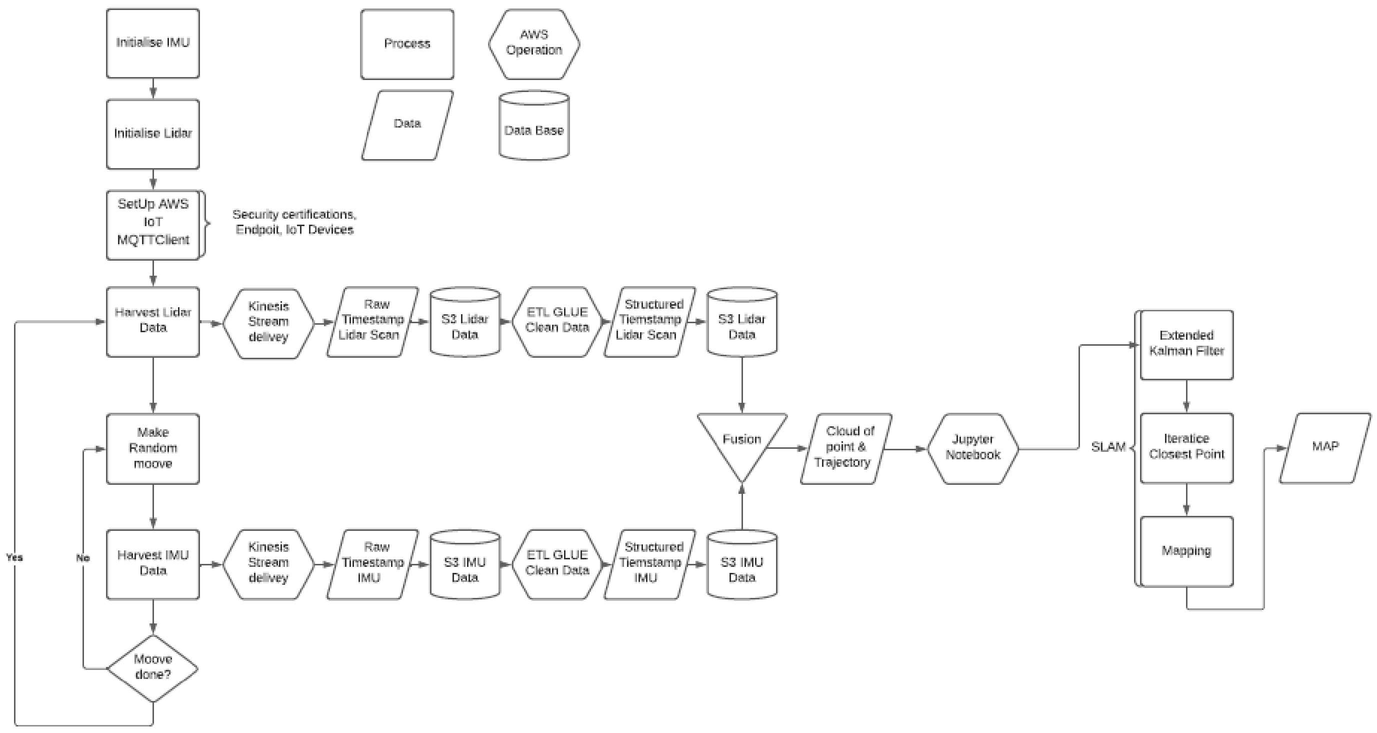


Figure 27 Data Cloud architecture of this project

First both the inertial sensor and the Lidar are initialised and connected to the AWS IoT MQTT Client. Then the Lidar record a first scan of data. Those data published to the IoT topic are delivered using the pipes of the stream delivery system of AWS Kinesis Firehose in a first folder of the S3 Lidar data base. Those data are raw and need to be cleaned of unnecessaries elements and structured. Therefore, is used the AWS ETL Glue, this part uses a database crawler to look in the database for new data. Then Glue structures the raw data in times series in the JSON format and save them in a different folder with all the cleaned data. After the first scan the robot can start moving, during this motion, the inertial sensor data are recorded and sent in real time using exactly the same process than the Lidar but at a higher frequency. Indeed, the inertial sensor is sending multiple recording per seconds while one Lidar scan takes 2.7 seconds. Those data are then accessed from the database by the Jupyter Notebook. This is an instance of notebook that can compute Python code online. On this server a small Python script is converting the JSON data to a CSV file structured accordingly to the need of the SLAM method. Then the SLAM method is applied, in a first time the cloud of points and the trajectory goes through the extended Kalman filter to predict the most probable position of each point, correcting the sensor error. Then each of these successive points is associated to the previous version of itself using the ICP method. These succession of clouds of points with the trajectories can create a map using as global reference frame the starting position. The output is a 2-dimensional map describing the trajectory of the robot and the surrounding walls. The codes relative to this part can observed in the Git-Hub repository of this project [34].

8 Results

8.1 Prototype

The first result of this project is very concrete and can be observed easily, this is the robot prototype. Able to move in space, to harvest the data of multiple sensors and to treat some parts of the information. This part is a real success as the hardware requirement are exactly the ones expressed in the project proposal. In a first time the robot was assembled using 3D printed parts, but it was considered to big and not relevant enough to be continued. Some improvements are still possible, as the Thymio II is not the best platform and could be upgraded to a better robot. The physical connection of the Thymio is not very stable and tent to disconnect, this is problematic because when the connection is interrupted the robot only executes the last given order. The result is that often the robot is going straight and ending in wall or going endlessly in a circle.
This 2D Lidar is a very good tool but it lacks the possibility to record the values while moving this would be a big improvement. The 9-axis inertial sensor is a very good device perfectly recording the trajectory in real time on this point everything is fine. To have a real computer like the Raspberry Pi 4 is very handy, as many modifications can be done directly on the robot without needing to pass through virtualisation portal like Secure Shell (SSH).

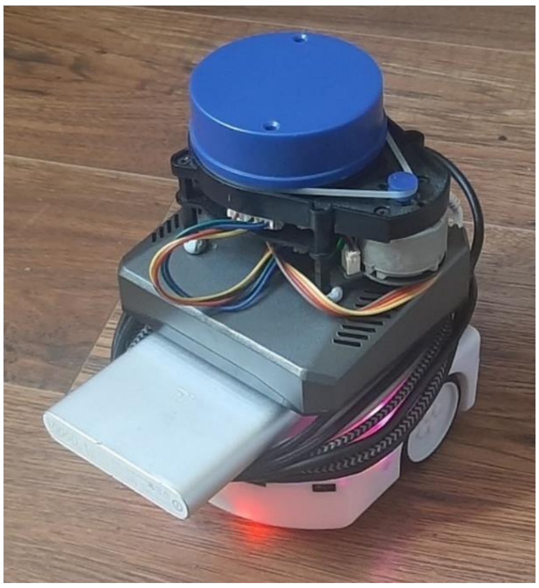


Figure 28 Project robot prototype

8.2 Cloud Architecture

The cloud architecture of the project was also a success. The data of both sensors were sent in real time to the different AWS services. Using the sensors as IoT really ease the process and can be adapted to any type of sensor. The values were published instantly (less than 0.1 second) to the IoT topic. The delivery stream and storing was a bit slower but still very acceptable (1-5 second). The slower was the cleaning and formatting of the data using ETL Glue (usually 56 seconds), it could be optimised or even this part could be omitted if the data were pre-processed directly on the Raspberry Pi. This technique of pre-processing the data before sending the data of the IoT is being developed nowadays with some time a small machine learning model present inside the IoT (previous internship of the author). This ease two things: there is considerably less data to send over a network which is most of the time constrained in delivery volume, this shorten the cloud processing time as the data are already clean when they arrive on the server. In the last version of the cloud computing the pre-processing was sometime omitted, the results weren't as good but still acceptable and significantly faster. This balance between quality of the data and speed of processing is key in this cloud computed architectures and need to be adjusted accordingly to the needs of the project.

8.3 Map

The map creation was the toughest task, but some quite satisfactory results were obtained. The trajectory was recorded as well as the cloud of points. The accuracy could be improved. It can be observed in the figure 29 the approximation of the map. There is always noise for a human eye the shape of the room is quite clear but the same for a reinforcement learning model is not sure. This is the biggest disappointment of the project to have a map with only the random trajectory of the robot. The robot is moving 10 cm and then turning at a random angle. It should have started in a random way and then learn how to discover efficiently. This result wasn't obtained in this project but the research and the methodology to finalise this work will be analysed in the next part.

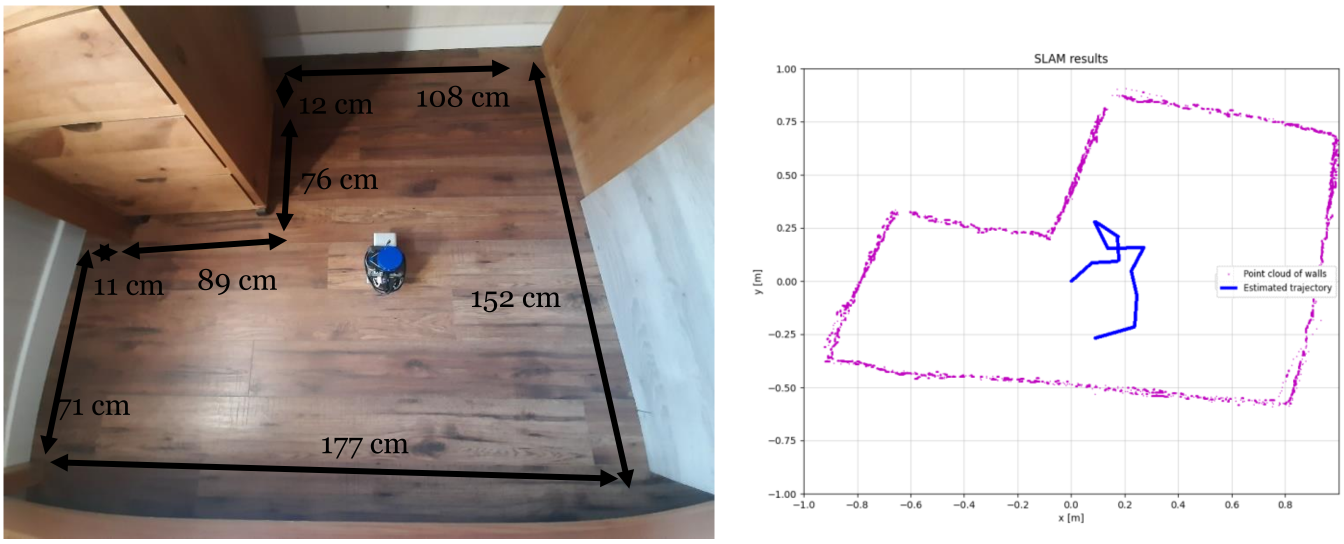


Figure 29 Comparison between real life environment with dimension on the left and obtained map on the right

Analysing the cloud of points made by the SLAM method, it can be observed that the walls which are straight lines in the real world are here only approximatively straight. It can also be observed that some distortion on the angles of the wall appear on the map. There are three probable combined causes, first the returning of the position by the inertial sensor suffers a slight shift over time, second the ICP continue to associate points to there supposed previous position. Finally, the noise of the sensor is too important and the Lidar is not perfectly levelled creating a distortion of the image. This resulting in a wrong angle and some bended wall. Overall, the shape of the environment is recognisable, but some work needs to be done to increase the quality of the map creation.

To improve the map creation first, more data were needed to be harvested. Here the physical constrains of the prototype are limiting the time of run, but if more iterations could be obtained the quality of the map would greatly increase. Second the correction of the position is good with the extended Kalman filter, but various method can be applied when some parts of the environment are known to correct the position. Finally, the linearisation of the surfaces could be done to obtain straight walls, for example feature based scan matching [38].

9 Further work

After a month trying to figure out how to make the reinforcement model work on practice some conclusions were made: the author doesn't have neither the technical skills nor the time to build a full reinforcement learning model and deploy it in a master thesis. Nevertheless, a lot of knowledge on this topic was learnt during this part. This part will explain the further work that could be accomplished to realise a full reinforcement learning. This part presents the strategies and the choice made during this study to build the model even if no relevant results were obtained. In a first time will be explained how the training should be conducted. Then how the machine learning model are chosen as well as its reward function. Finally, will be detailed how the inference of this model can teach the robot how to move autonomously into space.

9.1 Training

To practice the creation of a reinforcement learning model the DeepRacer tool from AWS was used. On this platform the training of an agent (a car) can be simulated, on an environment (a circuit) using stereo camera as the main sensor to recognise this environment. This platform is stuffed with examples to treat the received data with the camera, examples of reward function or examples of training model. Here it can customise premade model or create new one from scratch. The goal here is to make the car race the circuit as fast as possible without any human intervention and prior knowledge of the racing track. The goal of this master thesis is slightly different, but this is still a solid base to experiment the creation of reinforcement model learning. The first difference will be the sensors: in DeepRacer this is a stereo camera and in this project a Lidar coupled with a 9-axis inertial sensor. The environment is also different: a circuit in one case and in the other one a room. Finally, the goal is also different the goal of the race car is to do the shortest lap possible while the exploration robot aims to discover the room by doing the least steps possible.

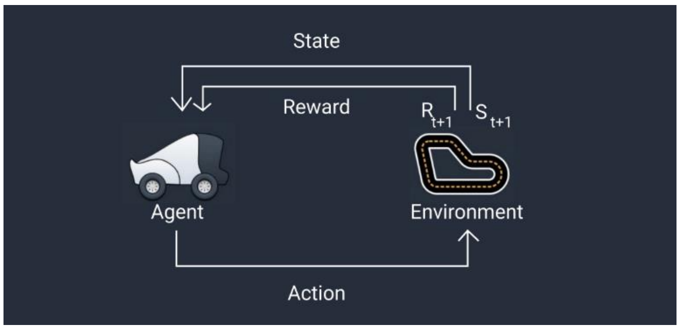
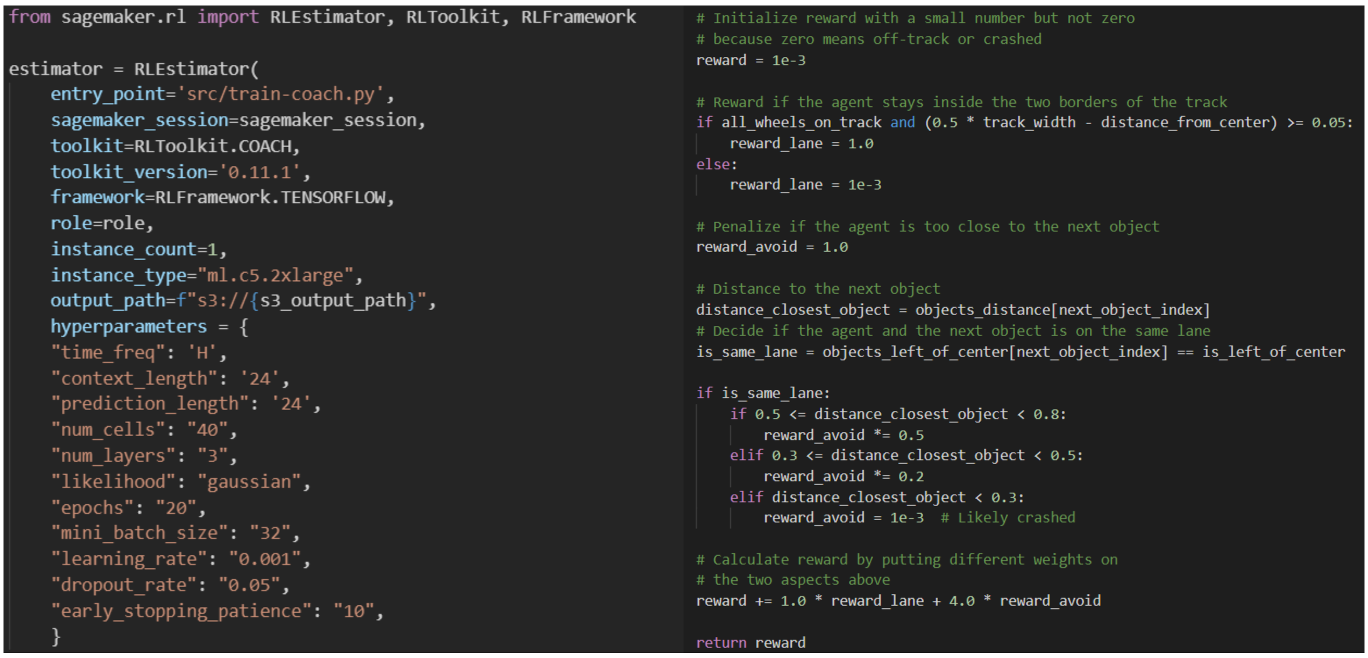


Figure 30 Reinforcement Learning DeepRacer [3]

9.2 Choice of the model

To choose the appropriate model the problem needs to be precisely defined as explained in the previous part. While practising on DeepRacer, the choice landed on RLEstimator. This is a good choice to build efficient reinforcement learning model. This is an SDK (Software Development Kit) packed with various tools, frameworks, reward functions, models and iterate them on AWS Sagemaker. This constructor takes argument like the choice of the instance (the volume of cloud computing power), the IAM role (to give access to different AWS services), some toolkit, some frameworks. It's also possible to choose already trained model if they correspond to the desired area of research. The RLEstimator needs to be provided with the entry point: the path to the Python file containing all the information about the environment (the data of the sensors). This model would also be chosen for this project as the constructor is perfectly adapted to the need of this project. The modification that needs to be done and that was one of the failures of this project is to adapt the entry point with the file adapting to the environment. Indeed, creating a new model from a model based on simulation data of a stereo camera to a model based real data form a Lidar + inertial sensor was a task that required quite some time. The other specificities shown in figure 31 such as the instance type or the various parameter would have been tuned to find the optimal result.



```python
from sagemaker.rl import RLEstimator, RLToolkit, RLFramework

estimator = RLEstimator(
    entry_point='src/train-coach.py',
    sagemaker_session=sagemaker_session,
    toolkit=RLToolkit.COACH,
    toolkit_version='0.11.1',
    framework=RLFramework.TENSORFLOW,
    role=role,
    instance_count=1,
    instance_type="ml.c5.2xlarge",
    output_path=f"s3://{s3_output_path}",
    hyperparameters = {
    "time_freq": 'H',
    "context_length": '24',
    "prediction_length": '24',
    "num_cells": "40",
    "num_layers": "3",
    "likelihood": "gaussian",
    "epochs": "20",
    "mini_batch_size": "32",
    "learning_rate": "0.001",
    "dropout_rate": "0.05",
    "early_stopping_patience": "10",
    }
```

```python
# Initialize reward with a small number but not zero
# because zero means off-track or crashed
reward = 1e-3

# Reward if the agent stays inside the two borders of the track
if all_wheels_on_track and (0.5 * track_width - distance_from_center) >= 0.05:
    reward_lane = 1.0
else:
    reward_lane = 1e-3

# Penalize if the agent is too close to the next object
reward_avoid = 1.0

# Distance to the next object
distance_closest_object = objects_distance[next_object_index]
# Decide if the agent and the next object is on the same lane
is_same_lane = objects_left_of_center[next_object_index] == is_left_of_center

if is_same_lane:
    if 0.5 <= distance_closest_object < 0.8:
        reward_avoid *= 0.5
    elif 0.3 <= distance_closest_object < 0.5:
        reward_avoid *= 0.2
    elif distance_closest_object < 0.3:
        reward_avoid = 1e-3  # Likely crashed

# Calculate reward by putting different weights on
# the two aspects above
reward += 1.0 * reward_lane + 4.0 * reward_avoid

return reward
```

Figure 31 RLEstimator specificities, Reward function example from AWS

9.3 Reward function

A reward function is a function assessing the behaviour of the robot and rewarding a score to this specific behaviour. The goal is to keep only the good behaviour by giving a high score (reward=1) to a good behaviour, for example discovering the full area in less than 10 steps. On the other hand, if the robot discovered in more than a 100 steps score will be low (reward=0.001), if the robot crashes the score is 0. The goal of the reward function is to obtain a maximal score over a given period of time. An example of a reward function is presented in figure 31, this function is aiming at avoiding obstacles and staying on track. It can be observed how the score is calculated and how different factors can be favoured.

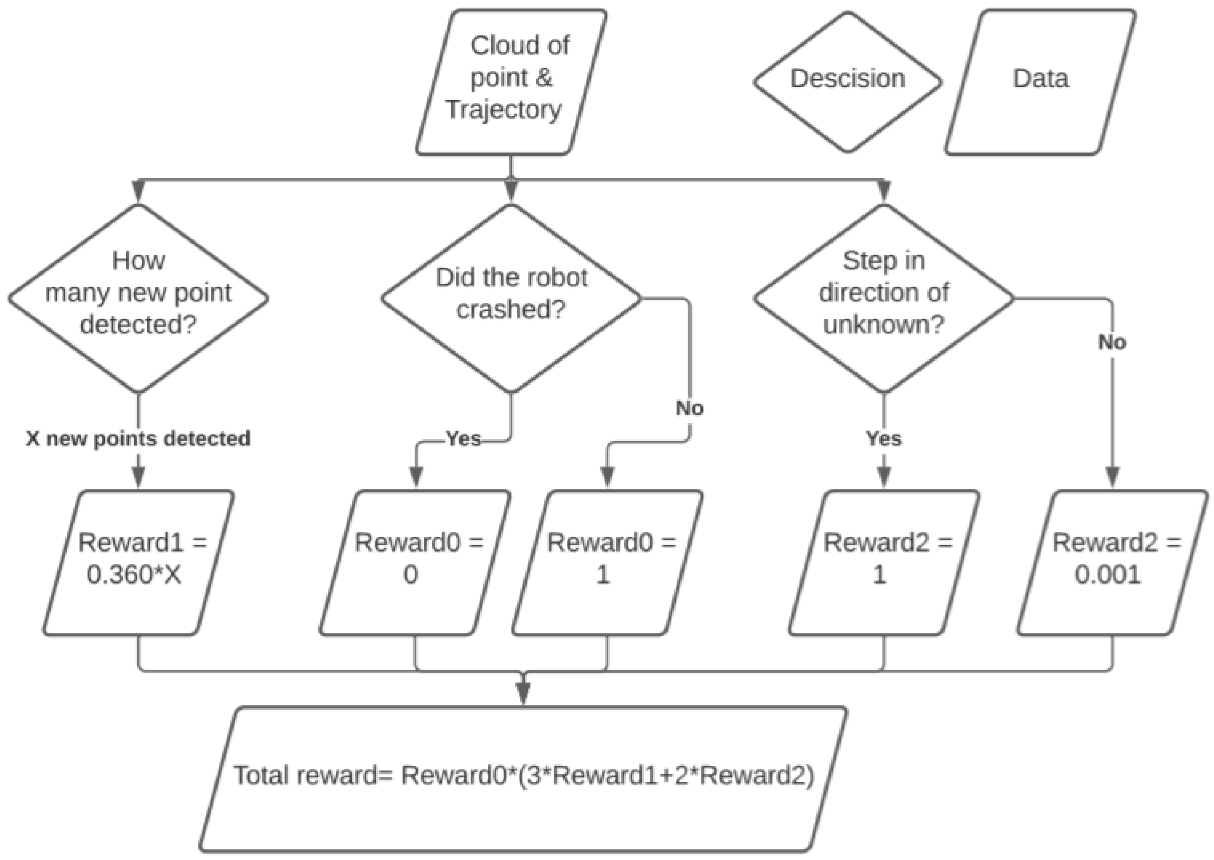
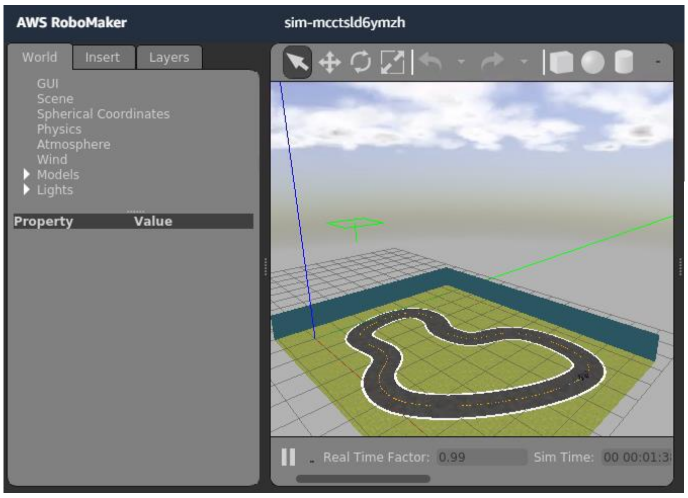


Figure 32 Reward function flowchart

In the case of this study the goal is to discover the room by doing the least number of steps. At each steps a Lidar scan is realised. This scan is compared to the previous one using the ICP method, and each corresponding point are associated. One way to build a reward function corresponding to this use case would be to encourage the discovery of as much new points as possible in a first time. The Lidar doesn't assign points if the distance is too big, this leaves a hole on the map for undiscovered points. The reward function could also reward the robot for each step taken in the direction of an undiscovered area. A basic principle in reinforcement learning is that the more precise the encouraged behaviour is, the faster the training will be, but the narrower the application will be too. On the other hand, the vaguer the rule is, the longer the training is and the wider the application is. A balance needs to be found between the limited computing power and the adaptability of the trained model. Reward functions are highly customizable to find the desired output. The figure 32 shows how could be constructed the reward function for the reinforcement learning of this project.

9.4 Inference

The inference is when the model is created accordingly to the environment with a good reward function and the real training can start. In this part the robot will start the recording of its movement (either in the simulation either in real life) and at the end of each step be rewarded for the action. In the DeepRacer training the car will start to move randomly on the track. Then after crashing, following the line, moving on the track or outside the track, the robot will start to understand what are the good and the bad actions rewarded by the reward function. The longer the training of the model is and the more adapted the reward function is, the higher score of the robot will be. When the score is decided to be sufficient, the training can be stopped. Here it is important to stop at the right moment: if too soon the model won't be precise enough, if too late the model can be overfitted. It means the robot is too dependent of this specific environment. Therefore, it won't be as efficient in new environments. When the model is trained, it can be evaluated. In this part the model will be faced with new data to determine if the training was of good quality or not. The data need to be different from the training data to avoid a biased overfitted model only trained for a specific environment. The evaluation will estimate the percentage of accuracy of the model, the elapsed time to complete the task and various other factors. This part is crucial to determine the quality of this model.



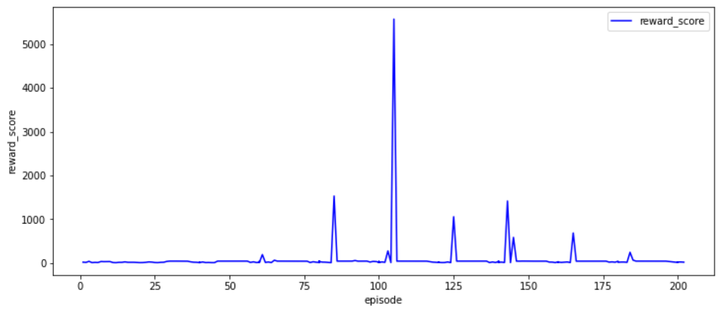


Figure 33 Result of the reinforcement learning practice on DeepRacer

For the studied project the training will be probably long as the environment is not simulated, and the raw data are always more complex to treat even when using an EKF. Another problem will be faced, the real time communication of the IoT sensors. Real time is always subjective and the communication of the data from the sensor to the server might need to be improved. Then the signal processing of the EKF, the ICP algorithm also takes time. All these steps need to be completed for a single reinforcement learning step. This delay needs to be optimised to have an acceptable training. The reward function and the hyperparameters also need to be tunned to create the most optimised training. Overall, this part will probably be the lengthiest, between adapting the entry point and all the training this project would need at least two more month to be fully conducted. The target was very optimistic but allowed to discover a wide field of research at the cutting edge of the machine learning and autonomous vehicle using cloud computed services.

10 Conclusion

In this project the prototype of an exploration robot was created. It can map a 2D indoor environment using SLAM technics combine with cloud computed tools. The robot was composed of a 2D Lidar and a 9-axis inertial sensor, those two combined using sensor fusion were processed using an extended Kalman filter to predict better result than the physical sensor. The cleaned data was then treated for each iteration of the recording by an ICP algorithm in order to create a 2D map of the environment. This process was executed in real-time communication over the cloud computing platform: AWS. The sensors were treated as IoT devices to constantly communicate information to the server, using various services the data where transformed, cleaned, and stored online.

The obtained results were a working prototype able to navigate in an environment either randomly piloted, with predetermined path or at random. A working cloud architecture was built to handle a big quantity of data in real time. A 2D map of a small environment was created with sufficient graphical detail for a human eye.

One of the objectives of the project wasn't completed, the machine learning part. The goal was to create a reinforcement learning model able to learn how to discover efficiently the environment. Using Sagemaker it was tried to create a notebook where all the mapping data could be used. Then using the RLEstimator with an adequate reward function, the model could have been iteratively trained. Due to a lack of time and technical skills, this part will be moved to the further work of this project.

References

[1] Mars Curiosity Rover, NASA, https://mars.nasa.gov/msl/home/ (Accessed 28/07/21)

[2] REMUS 600, Kongsberg, https://www.kongsberg.com/globalassets/maritime/km-products/product-documents/remus-600 (Accessed 28/07/21)

[3] A. Draou, "Electronic differential speed control for two in-wheels motors drive vehicle," 4th International Conference on Power Engineering, Energy and Electrical Drives, 2013, pp. 764-769, doi: 10.1109/PowerEng.2013.6635706.

[4] Yousif, K., Bab-Hadiashar, A. & Hoseinnezhad, R. An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics. Intell Ind Syst 1, 289–311 (2015). https://doi.org/10.1007/s40903-015-0032-7

[5] Cho, BS., Moon, Ws., Seo, WJ. et al. A dead reckoning localization system for mobile robots using inertial sensors and wheel revolution encoding. J Mech Sci Technol 25, 2907–2917 (2011). https://doi.org/10.1007/s12206-011-0805-1

[6] Y. Li et al., "Deep Learning for Lidar Point Clouds in Autonomous Driving: A Review," in IEEE Transactions on Neural Networks and Learning Systems, doi: 10.1109/TNNLS.2020.3015992.

[7] J. Xu, “Accuracy Estimation of 2D SLAM Using Sensor Fusion of Lidar and INS”, University of California, Berkeley, 2019.

[8] S. Y. Chen, "Kalman Filter for Robot Vision: A Survey," in IEEE Transactions on Industrial Electronics, vol. 59, no. 11, pp. 4409-4420, Nov. 2012, doi: 10.1109/TIE.2011.2162714.

[9] Choi, KS., Lee, SG. Enhanced SLAM for a mobile robot using extended Kalman Filter and neural networks. Int. J. Precis. Eng. Manuf. 11, 255–264 (2010). https://doi.org/10.1007/s12541-010-0029-9

[10] F. Zhang, S. Li, S. Yuan, E. Sun and L. Zhao, "Algorithms analysis of mobile robot SLAM based on Kalman and particle filter," 2017 9th International Conference on Modelling, Identification and Control (ICMIC), Kunming, China, 2017, pp. 1050-1055, doi: 10.1109/ICMIC.2017.8321612.

[11] K. Ayush and N. K. Agarwal, "Real time visual SLAM using cloud computing," 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT), Tiruchengode, India, 2013, pp. 1-7, doi: 10.1109/ICCCNT.2013.6726744.

[12] DeepRacer, AWS, https://aws.amazon.com/fr/deepracer/ (Accessed 28/07/2021)

[13] A. Notsu, K. Yasuda, S. Ubukata and K. Honda, "Optimization of Learning Cycles in Online Reinforcement Learning Systems," 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan, 2018, pp. 3530-3534, doi: 10.1109/SMC.2018.00597.

[14] J. K. Makhubela, T. Zuva and O. Y. Agunbiade, "A Review on Vision Simultaneous Localization and Mapping (VSLAM)," 2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC), Mon Tresor, Mauritius, 2018, pp. 1-5, doi: 10.1109/ICONIC.2018.8601227.

[15] P. Shengguang, "Overview of Meta-Reinforcement Learning Research," 2020 2nd International Conference on Information Technology and Computer Application (ITCA), 2020, pp. 54-57, doi: 10.1109/ITCA52113.2020.00019.

[16] N. Touzin, “OPTIMAL STOCHASTIC CONTROL, STOCHASTIC TARGET PROBLEMS, AND BACKWARD SDE”, Ecole Polytechnique, Paris, Department de Mathématiques Appliquées, 2020

[17] B. Balaji et al., "DeepRacer: Autonomous Racing Platform for Experimentation with Sim2Real Reinforcement Learning," 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 2746-2754, doi: 10.1109/ICRA40945.2020.9197465.

[18] S. Saraswathi and A. Allirani, "Survey on image segmentation via clustering," 2013 International Conference on Information Communication and Embedded Systems (ICICES), 2013, pp. 331-335, doi: 10.1109/ICICES.2013.6508376.

[19] W. T. Beyene, "Reduced-order modeling of high-speed channels using machine learning techniques: Partitional and hierarchical clusterings," 2017 IEEE 26th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS), 2017, pp. 1-3, doi: 10.1109/EPEPS.2017.8329767.

[20] M. Saito and J. Mitsugi, "High Resolution Time-of-Flight Measurement with Narrow-Band COTS Ultrasonic Transducers," 2018 IEEE SENSORS, 2018, pp. 1-4, doi: 10.1109/ICSENS.2018.8589571.

[21] I. Beavers, “Kalman or FIR Filter for My IMU?”, AnalogDialogue, 03/2016, https://www.analog.com/en/analog-dialogue/raqs/raq-issue-127.html#

[22] R. Tiar, M. Lakrouf and O. Azouaoui, "FAST ICP-SLAM for a bi-steerable mobile robot in large environments," 2015 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), 2015, pp. 1-6, doi: 10.1109/ECMSM.2015.7208683.

[23] Trackmania, https://www.ubisoft.com/fr-fr/game/trackmania/trackmania (Accessed 29/07/21)

[24] MPU-6050, https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf Datasheet, (Accessed 29/07/21)

[25] W. ZHANG and C. QI, "Pose Estimation by Key Points Registration in Point Cloud," 2019 3rd International Symposium on Autonomous Systems (ISAS), 2019, pp. 65-68, doi: 10.1109/ISASS.2019.8757773.

[26] MPU-9250 https://invensense.tdk.com/download-pdf/mpu-9250-datasheet/ (Accessed 29/07/21)

[27] HC-SR04 https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf (Accessed 29/07/21)

[28] Thymio II https://www.thymio.org (Accessed 29/07/21)

[29] Raspberry Pi 4 https://www.raspberrypi.org/products/raspberry-pi-4-model-b/ Accessed 29/07/21)

[30] J. Brownlee, “A Gentle Introduction to Concept Drift in Machine Learning”, Machine Learning Mastery, 10/12/2020.

[31] R. E. Kalman, “A New Approach to Linear Filtering and Prediction Problems” in Transactions of the ASME–Journal of Basic Engineering, 82(D), pp. 35-45, 1960.

[32] P. J. Besl & N. D. McKay, ”A method for registration of 3-D shapes” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2), pp. 239-256, 1992.

[33] Y. Tian, L. Deng and Q. Li, "A KNN Match Based Tracking-Learning-Detection Method with Adjustment of Surveyed Areas," 2017 13th International Conference on Computational Intelligence and Security (CIS), 2017, pp. 447-451, doi: 10.1109/CIS.2017.00104.

[34] T. A-Milhomme, ‘Smartly discover environment using IoT sensors with cloud computed machine learning’ https://github.com/TimMilhomme/Smartly-discover-environment-using-IoT-sensors-with-cloud-computed-machine-learning- (Accessed 11/08/2021)

[35] Yosh, “A.I. teaches itself to drive in Trackmania”, https://youtu.be/a8Bo2DHrrow (Accessed 13/08/2021)

[36] YDLidar X4, Data sheet, https://www.ydLidar.com/Public/upload/files/2021-08-10/YDLIDAR%20X4%20Data%20sheet%20V2.0.pdf (Accessed 13/08/2021)

[37] J. Engel et al., “Large-Scale Direct SLAM with Stereo Cameras”, IROS 2015, Hamburg

[38] Z. He, X. Wang, J. Liu, J. Sun and G. Cui, "Feature-to-Feature Based Laser Scan Matching for Pallet Recognition," 2010 International Conference on Measuring Technology and Mechatronics Automation, 2010, pp. 260-263, doi: 10.1109/ICMTMA.2010.464.

[39] PyLidar3 library, https://github.com/lakshmanmallidi/PyLidar3 (Accessed 19/08/2021)

[40] Pandas library, https://pandas.pydata.org (Accessed 19/08/2021)

[41] “MQTT: The Standard for IoT Messaging”, https://mqtt.org (Accessed 19/08/2021)

[42] AWS IoT Core https://aws.amazon.com/iot-core/ (Accessed 19/08/2021)

[43] JSON format, https://www.json.org/json-en.html (Accessed 19/08/2021)

[44] Apache Parquet, https://parquet.apache.org/documentation/latest/ (Accessed 19/08/2021)

[45] AWS Glue, https://aws.amazon.com/glue/ (Accessed 19/08/2021)