Abstract

Swarm Robotics is the branch of engineering that studies the development and interaction of multi robot systems coordinated in a decentralised form. Despite an increase in works around this subject in the past decades, there is still much to do specially in the area related to the competence between groups of robots. Moreover, the implementation on real situations like resources gathering is missing as most investigations focuse on abstract formations.

The research done aims to fill this gap developping two swarm teams placed in a unknown maze that explores the room and gathers resources using designed algorithms. A simulation environment using Gazebo has been used to develop the swarm robotic system which main method of interacting was Robotic Operating System. Furthermore, an adapted robotic model has been designed and built along with other three robots to show the hardware required to implement in the actual work the system and the difficulty when trying to implement work from the simulation into the real world.

The outcome has been a simulation and hardware model suitable for swarm robotics and a sfotware system for coordinating robots to perform resource gathering and environment exploration that improves the results from individual robots. Moreover, a general analysis over competence in swarm has been reviewed when performing the work.

Acknoledgements

I would like to thank my supervissor Ali Syed for the support and advise given and to the member of staff Andrew Bilbrough for helping me in the laboratory when needed and for giving me the components I was ordering efficiently. A special thanks as well to Dr. Mehmet Dogar for providing advice in the methodologies related to robot exploration and mapping.

I would also like to show my gratitude to all those engineers and programmers that back in 1991 helped to create one of the greatest achievements for humanity allowing everyone to share their knoledge and to take part in one of the greatest momemnts of human species.

Table of contents

Chapter 1: Introduction

1.1 Motivation

Robotics is an emerging field in engineering that is becoming increasingly relevant to humanity. In the incoming years, thanks to the advance in maths, physics, and computer science, it is expected to advance more than ever. The relevance of this branch of engineering along with its incoming development were strong reasons to choose a research related to it.

The work presented in this report focuses in Swarm Robotics (S.R), branch of robotics that focuses on recreating characteristics of social animals, specially cooperation, into robotics. It is based on developing multi robot systems that benefits from communication, cooperation and collaboration between robots to grant them to perform tasks that could not be achieved by individuals.

This filed has a significant relevance in mechatronics (mechanical, electrical and computer science subjects combined) and robotics and due to its many applications for humans, it was thought to be one of the best choices to work at. It has the potential to provide solution to many problems in robotics and to give a better understanding of human race as well as social animals. Moreover, as recent researches have stated, sociability might be the necessary characteristic to develop consciousness. To study S.R can give a better understanding of this important question. [1.1]

The list showed below is a general overview of some of the many applications of the subject. Any of them a strong statement to investigate on this area:

- **Industry (secondary sector):** This is one of the fields that will most benefit from S.R. From organizing big warehouses like Amazon robots already do with AI to coordinate assembly and manufacturing process, the introduction of it in industry will help companies to reduce costs and risks and to increase production and quality.

- **Primary sector:** Automatization of agriculture already started but the robots get into the work the most necessary will be to equip them with some sort of AI and S.R intelligence to achieve optimization. This might be beneficial to poor regions where if robots get cheaper in the incoming years, they will help to reduce cost, effort and time to provide food and water.

- **Tertiary sector (services):** Areas that can benefit are medicine, using nanobots synchronised or operation rooms with different machines coordinated [1.2], in transport service with automated drones or cars for delivering packages in an optimal or even for vigilance using small swarms to guard a place.

- **Rescue and Exploration:** These are the subjects of the work done. To use a series of simple coordinated robots could improve a non-human exploration of dangerous regions that cannot be reached by any humans. In natural disasters, swarms of drones could improve fire extinction as well as people or natural resources collecting. Resource gathering is the main subject of the research.

1.2 Problem Statement

This project aims to create a multi robot coordinated system and algorithms for environment exploration and resource gathering, to clarify how competence in S.R might influence two organised groups of robots (barely explored by researches on S.R) and to analyse the development of the hardware and software design and development for the multi robot system analysing the best design decisions and the best software methods to be implemented.

1.3 Project Objectives

The project consists in two teams of two simple homogeneous robots deployed in an unknown maze. Each team has a headquarter where to take the resources found. These are represented as light spots. Every time a robot detects one by the luminosity increase, it saves its position and goes back to its base reproducing the collection and storage process. The competence between teams is generated by running them simultaneously.

One robot maps first, the environment and later both explore it. Moreover, they use a designed algorithm for optimising the collection by coordinating in an intelligent way.

The final goal is to provide a solid system for a multi agent group of robots that coordinates the exploration of an unknown space as well as to optimise resources pursue.

The robots fulfil the main characteristics of S.R. They are homogeneous, simple (to achieve complex tasks, they need cooperation for making their individual intelligence bigger as a collective one), cheap, small-sized and autonomous. They also are able to sense their surroundings, localise themselves (odometry system) and communicate between the team members.

The project is divided in two core parts: the simulation and the real-life implementation. The first one is the main section to achieve as it is where all the S.R software and system is developed. It brings the possibility of testing the project with a greater number of robots than the budget lets in real life and to test them in different environments and with different types of components. It is where the exploration and the resource gathering algorithms are tested. The real-life implementation involves hardware for building the four robots from scratch and to develop a design that can potentially perform the simulation.

Chapter 2: Background

This chapter covers the preliminary research made in the field and aims to provide some introductory background on the topic.

2.1 Literature Review

S.R is the field that studies multi robot systems that follows its own principle. The principal areas that forms one of these S.R systems are communications, swarm intelligence, stigmergy and the coordination algorithms. It concrete, S.R studies how a group of individuals and their behaviours are more robust and capable than a single robot in doing tasks. The subject is inspired by nature examples of social animals that collaborate in societies like ant's colonies, humans, bees and so on.

Swarm intelligence studies started on the decade of the 80's realising that an emerging group intelligence can be obtained also in robotics. Rather than focusing on the AI or characteristics of a single robot, it works with group behaviours and algorithms based on a decentralised strategy with local rules and exchange of information and where no individual has access to all the information. [2.1]

These are the most relevant S.R principles [2.2]:

Formation	Cluster of individuals.
Appearance	Set of homogeneous robots.
Network Structure	Decentralised and self-organised.
Communication	Locally between individuals or through the environment.
Scalability	Yes.
Reparability	Fault of tolerance. As it is a decentralised homogenous group, there is no fundamental individual and they can be easily replaced or readjusted with no harm for the cluster.
Task Management	S.R works with parallel actions or specialization. Groups or individuals can do different tasks simultaneously by specialization.
Price	Normally cheaper than a single more complex robot and more energy efficient.

Moreover, there are some relevant concepts related to the topic to bear in mind:

Stigmergy: Capability of swarm groups of working in one tasks without communicating. For instance, if a robot starts a task with an action, afterwards, just because of the creation of this process another robot keeps or starts working on this task. This leads to a decentralised behaviour focused on constantly improving the realisation of the task by individuals of the group. [2.3]

Artificial Culture: Phenomenon of teaching and variating sets of behaviours between individuals of the same or different groups of swarms. The similitude with culture and transferring traditions through generations is the responsible of the name of this effect. Despite being studied with a group that cooperates, it has not been analysed how two competing groups can influence the behaviours of each other. [2.4]

Moreover, all S.R are considered to develop these fundamental features: communication system, navigation, obstacle avoidance, localisation, search and coordination.

2.2 Methodology

This part describes some concepts for the methods employed on the simulation to provide a better understanding of the section concerning to it. In particular, it refers to the naming used in ROS (Robotics Operating Systems).

ROS and the simulator have both common concepts that are explored later on this report. This is the background on the namings employed in Chapter 3:

- **Node:** executable that uses ROS to communicate with other nodes using topics and services. It is the minimum element of a ROS system. Each component like a camera or a sensor are attached to one node if they need to communicate with other robots or with different parts of the same bot. [2.5]

- **Topic:** mean where the nodes send and receive messages between each other providing an unidirectional streaming communication. However, they are not aware who they are communicating with. [2.6]

- **Message:** "simple data structure, comprising typed fields." [2.7]. It can be of diverse data types.

- **Plugin:** Gazebo classes created for increasing the functionality of the desired model. They also work with ROS. [2.8]

- **Transformation Frames:** 3D frames that represents all the coordinate frames from the components of the robot models and the world. Transformation Frames methods allows to compute the transformation and conversion from one to another. [2.9]

- **Broadcaster:** method for frame transformation or for publishing a new virtual or real frame.

2.3 State of the Art

S.R is a field that has been experimenting during these past decades a growth due to its still short life as a branch of robotics. It started with the academic work of Swarm-bots (controlled by European Commission) at the beginning of millennium. [2.10]

From there, several investigations and researches have been done on the field, focused specially in theoretical studies or practical situation that have not been brought

yet to the market. An outstanding work by University of Harvard was the most “mediatic” one until the date as it had up to 1000 Kilobots [2.11]

Moreover, some attempts in industrialising and commercialising them have been explored. In this section is important to point out the work done by team K that distribute several models including Kilobot (this robot actually uses some of the software used for the project and its MCU is very similar to the one used.).[2.12]

Universities have been taking an important position on the research of S.R. Famous technological universities like Sandford with the creation of Zooids and Universities in this country like Bristol [2.13], Sheffield [2.14] or York [2.15] have been the main font of investigation.

Chapter 3: Simulation

3.1 Overview & Multi Robot Model

The core investigation of this project was performed in a robotic simulator using the methods that can be applied in the real robots. Therefore, the first milestone was to pick the proper software workspace that would make possible a realistic representation.

Firstly, Buzz [3.1] was considered as the S.R programming language and ArGos [3.2] as the simulator as they were originally created for this branch of robotics. However, several inconvenient were found when trying to use these software that led to pick different ones.

The main reason was that they were not up to date as well as difficult to install and use. Scrimmage [3.3] was another inspiring simulator considered but it was still lacking some requirements. These specifications were the existence of a reliable source of tutorials, an active community around the platforms and a big team of active open source developers behind them. None of them fulfilled all these parameters. Furthermore, the fact that they were too specialised in S.R became a disadvantage instead of something beneficial as they were lacking of tools for recreating a realistic robot model or world.

Finally, two open source softwares with a large community of developers and professionals along with plenty tutorials were found. They were: ROS [3.4] (Robot Operating System) as the multi-agent multi robot standardised communication, Gazebo [3.5] as the simulator and Rviz [3.6] as the tool for visualising mapping, navigation and localization processes.

- **ROS (lunar version):** Brings a communication system between nodes, parts of the same or different machines that need to exchange information. This allows to not disturb the functionality of the robot if a node is changed and it allows to reproduce the same simulated results in the real world getting adapted to the real robot if correctly interfaced. Despite not being considered as a full operating system, it also provides abstraction on the software and hardware low levels.

- **Gazebo (using ROS plugins):** Robust realistic robotic simulator.

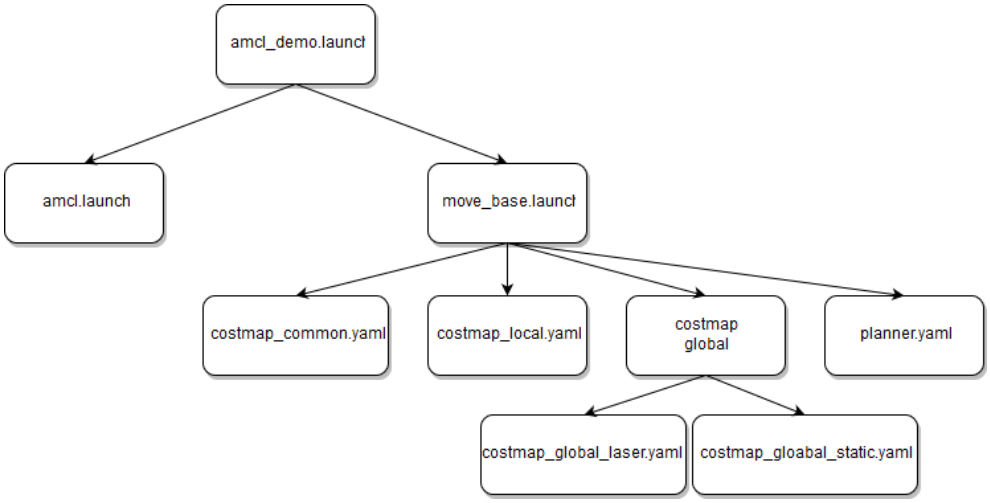
- **Rviz:** 3D visualiser of ROS data from topics and services.

These softwares run on Unix systems. As the computer used on this project was Windows, after trying its new subsystem for Linux using bash shells, it was found that an Ubuntu virtual machine would be easier and more adaptable in terms of CPU and memory consumption.

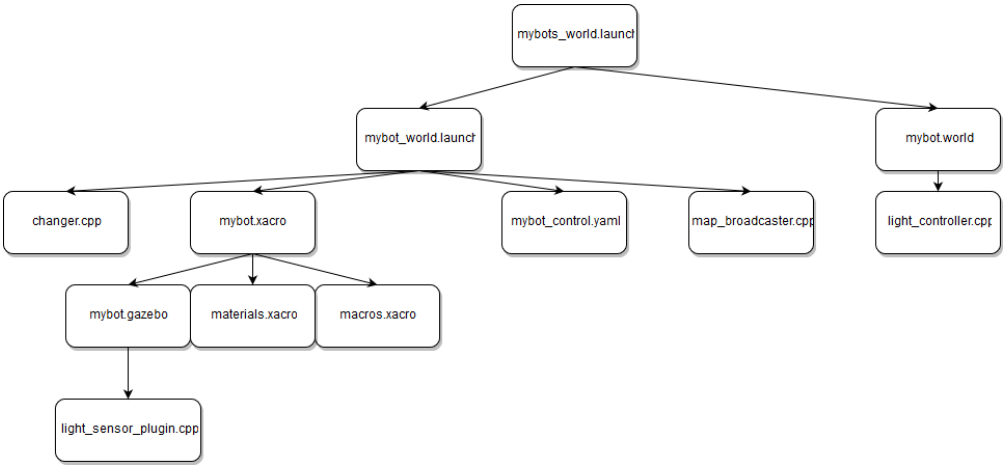
3.1.1 Code Scheme

Due to the amount of code file written and the different subsections of this chapter, it was considered to be a good idea to show a more general look of all of them and their inter relationships. These are the following:

These are the created files during the project for a working multi robot system. They will be described in the incoming sub sections:



These are the created files for allowing navigation and path planning. Despite not being coded all of them from scratch, all needed to be understood adequately for adapting their parameters to the used robot model and to the multi robot system.



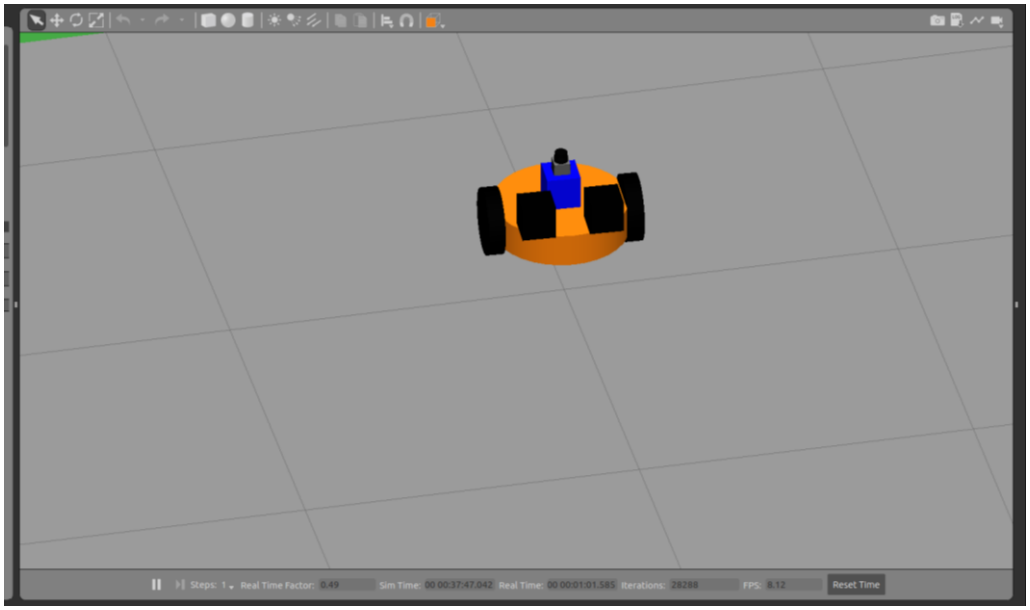
3.1.2 Robot Model

Once the work space was set up in its folder, the first step was to design a robot model that interfaces ROS and Gazebo and similar to its hardware version. The following packages were created for the model:

- **Robot Description:** It holds the robot definition both for ROS in form of URDF (Unified Robot Description Format) file, a Gazebo file for the model, materials and macro files. It contains the needed information of the model components (links) and relationships (joints).

- **Robot Control:** It has the YAML (Yet Another Mark-up Language) configuration file of the wheels drivers and joint states.

- **Robot Gazebo:** Contains all the launch files and the ". world" file (SDF (Simulation Description Format) code that has the world maze environment).



Simulated Robot Model

3.1.2.1 Robot Description

This section discusses the descriptive files for the robot model and justifies the simulation design compared with the real robot.

"*mybot.xacro*" is the main file that holds the definition of the model and includes the rest of the description documents. It uses one macro for the right and left wheels and another for the materials. ***The code created is shown in the Appendix [1] to give an overview on how these files are coded.*** The model links are intended to be as close as possible to the real robot. They are the following:

Component	Dimensions	Joints	Description
Chassis	10x40cm cylinder.	Parent of the rest of the links	Main body of the robot.
Caster Wheel	5cm sphere.	Children link from the chassis.	Placed centred at the front of the robot to give the necessary stability. In the real-world design, two caster wheels instead of one had to be used for ensuring stability. However, this does not make a difference from the simulated model to the real one.
Left/Right Wheels	5x10cm cylinder.	Children link from the chassis using a continuous joint (one degree of freedom).	Placed at the sides of the chassis. They use a simple transmission using an effort joint and setting motors as the actuator. The joint and wheel controller later uses this. They are defined in the "*macro.xacro"* file.
Sensor Base	10cm cube.	Children of the cassis and parent of "sonar sensor".	Placed at the centre top of the chassis, it holds the "sonar sensor" (represented as a laser scan sensor) and avoids that its rays get interfered by other robot part (like the light sensors).
Hokuyo Laser (acting as the "sonar sensor")	10cm cube.	Children of sensor base link.	As it did not exist any sonar sensor plugin or component available for ROS or Gazebo, a laser sensor was adapter to try to simulate it by changing its parameters. The result was in performance similar to have two sonar sensors. 180º was the range picked to have as it was easier and it avoids coordination issues related to publishing two sensor data to the same topic than placing two 90º ones simulating the two real sensors. The results would be the same as using two of them.
Light Sensors	10cm cubes.	Children of the chassis link.	Placed at the front top part of the chassis. Like in the previous case, there were no light sensor model available so a plugin using the recommendations from was created using camera sensors for determining the input luminescence of their image.

These components have a visual element for been displayed in Gazebo, an inertia component to reproduce their physics and a collision tag to interact with the world and other objects.

"*mybot.gazebo*" allows ROS URDF file to be used in Gazebo so that the model can be spawn in the simulation. It is where sensor and actuator properties are defined. These are:

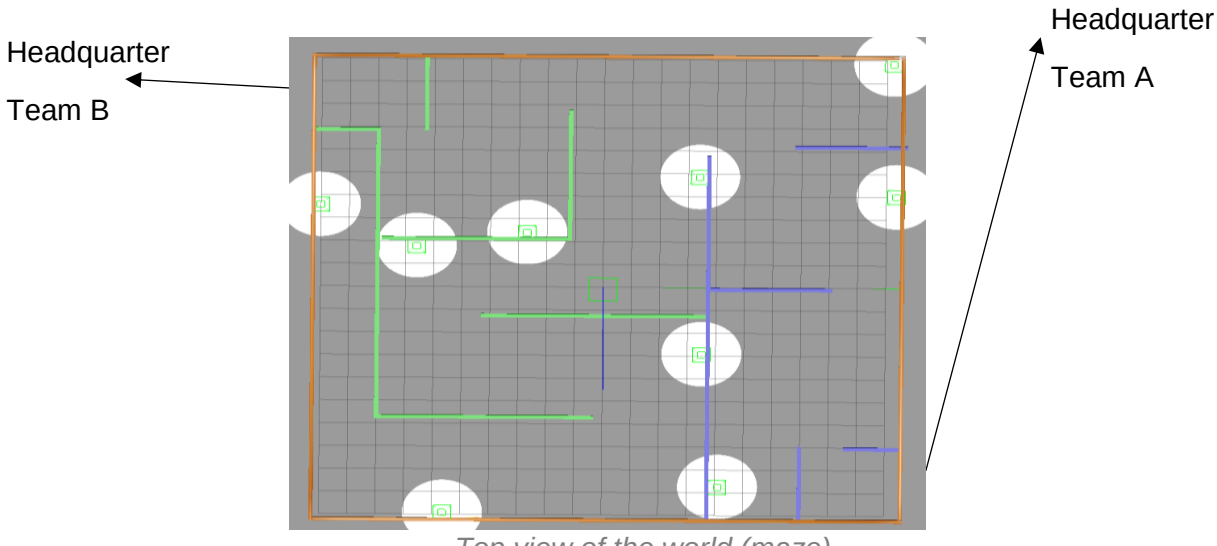
- **Differential Drive Controller:** It publishes odometry information of the robot (spatial information and dynamic data). In the real model this is performed by encoders but in the simulation is done using a diffrential drive plugin. Its input is the data from right and left wheels (link and joint information). It provides them a torque of 20Nm and the topics /cmd_vel (where velocity of the robot is published) and odom_diffdrive (where odometry data is published). Its base frame is the footprint, a link created to be the parent of the chassis and to allow the hole robot to move around the maze.

- **Hokuyo Link:** It defines the "sonar sensor" propertie and therefore,adapts it to look like a sonar sensor (although for time consuming, computing complexity and simplicty reasons some parameters were not modified). It has a 2D scan with 720 samples distributed over 180º, a range from 10cm-30m and a resolution of 1cm. It uses a gaussian type noise approximation. [3.7]

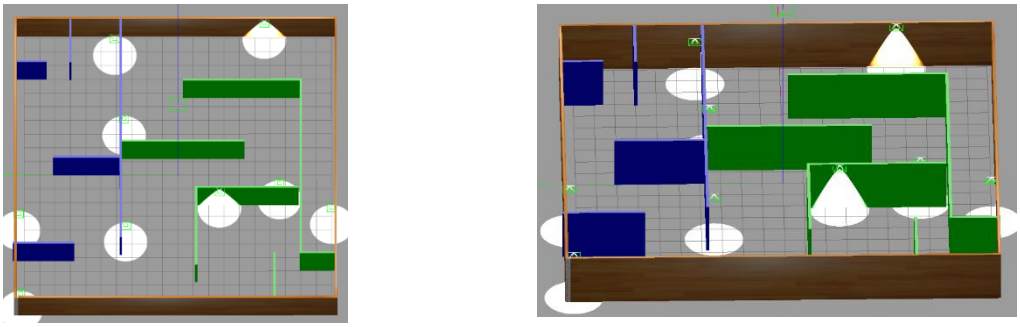
- **Light Sensor:** It defines the camera topics and parameters needed for using the created plugin and for determining the luminiscence from the image pixels.

3.1.2.2 Robot Gazebo

This package contains, apart from the launchers, the world simulated environment. It consists in a maze with two headquarters and 9 light spots that can be on/off representing the resources.These, are intended to be equally distributed over both sides of the maze to avoid benefit any team but also being an odd amount of them to facilitate competence. Gazebo editor was employed to generate the SDF file that defines it.



Top view of the world (maze)



Isometric views of the world (maze)

The maze was designed to make the robots face a difficult environment with many obstacles, narrow paths (the entry to each headquarter) and many corners to test if the obstacle avoidance and path planning algorithms were robust enough.

3.1.2.3 Robot Control

This package contains the YAML file that specifies the joint state controller and the joint effort controller for both right and left wheel.

3.1.3 Light Sensor Plugin

This section is a brief explanation on the implementation of the light sensor plugin created following the tutorial of [3.8]. This was adapted to the robot model used and to a multi robot system. A topic tag holder was added to the class code so that it can be specified later in "*mybot.gazebo*".

This plugin takes a camera sensor and adds up its RGB image values of each pixel. Despite theoretically not standing for illuminance, in practise the pixel intensity is related to it as the clearer an image is the brighter the environment was. Finally, it publishes a topic with illuminance message.

3.1.4 Light Plugin

This plugin class objective is to provide control to the spot lights around the simulated environment. It is a Gazebo plugin that subscribes to the robot's pose and light's pose and publishes to the light spot range Gazebo topic.

Lights are intended to be natural resources and therefore, they also sometimes appear and disappear from one spot. The code stores light and robot position in arrays and if the robots get too close to the light (within a squared range of 3m) they go off. Moreover, as the times passes, the lights start going off progressively in periods of 2 minutes leaving at the end only two spots on and after 8 minutes this cycle is reset.

This aims to increase competences between the teams making them gather for less amount of resources.

The range publishing is done by a method which periodically updates the range of each light source. Another critical issue to point out is that the pose topic message of Gazebo is in the form of a string so a function had to be used to get first the desired object name and then to convert its x and y coordinates into floats.

The reason of switching the lights off when a robot is too close is to represent the wear of the source spot.

3.1.5 Multi Robot Modelling & Launchers

This sub chapter explains how to implement a multi robot system from the same robot model. To summarise, the files previously described are modified to spawn several robots of the same model with their own topics, frames and components. This is fundamental to avoid overlapping of data from sensors, odometry and velocity and for displaying them and their frames. This was one of the most arduous process as the existing multi agent systems in ROS already had their own robots modelled and there is barely information on how to adapt it.

There are two main launchers used in the implementation of the multi robot system: the robot model and the final one that holds everything. These two launchers are showed into detail in Appendix [2,3]

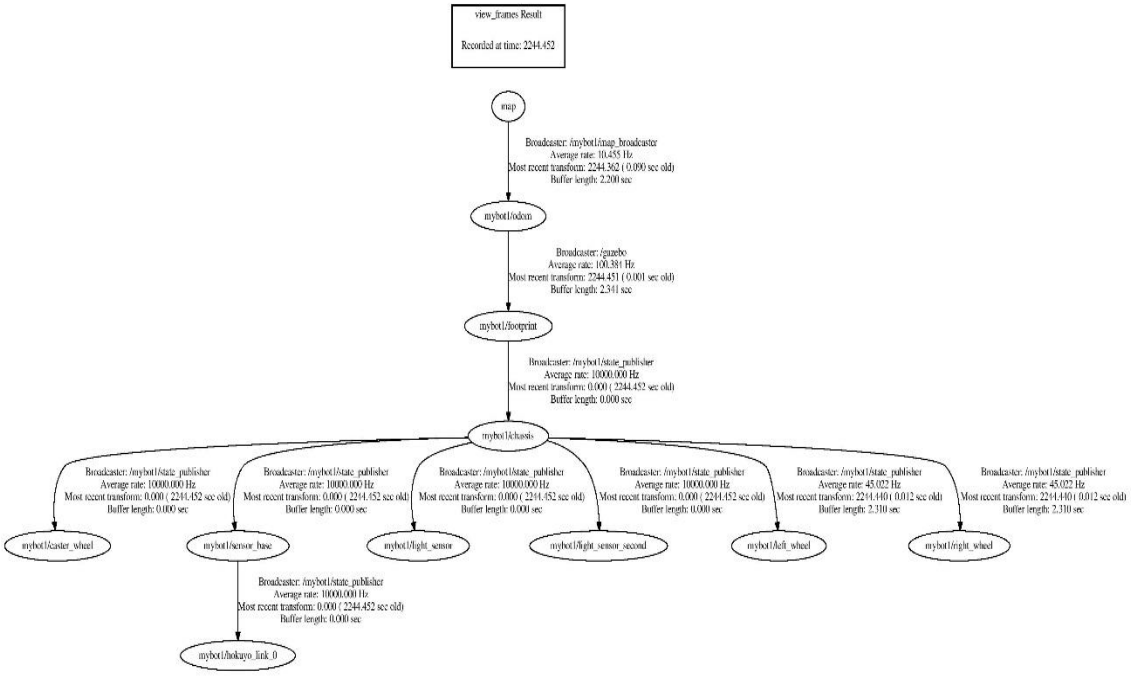
3.1.5.1 Robots & World Launcher

XML launch file that includes the Gazebo world, the four robots of the simulation and Rviz node.

Each robot is grouped in a “group” tag which takes as parameters a namespace (naming of the robot), its spawning position and includes the robot model launcher.

3.1.5.2 Robot Launcher

It includes the XACRO file holding the description of the robot. It also creates a spawn type node to place it in the specified location on the maze. Moreover, it has a node broadcasting a transformation between the “map” general frame created in a C++ file (it creates a “map” frame which is the common parent frame to all the robots) and the robot “odom”, parent frame of the robot footprint. Furthermore, another node publishes the joint states and remaps them to the appropriate unique individual robot name and another one that loads the controller of the right and left wheels. Below is the frame tree of all the robots with their frame broadcasters to give a better idea of the above explanation:



To use the model as a multi robot system, its description files needed to be changed to use a prefix of each robot's name for their frames and topics. All links and joints had to be prefixed with the robot namespace passing it as an argument from the launch file. For instance, if the body was "*chassis*", it would have to be renamed as *"$(arg nsp)/chassis*" where *"$(arg nsp)*" was the unique robot name.

In "*mybot.gazebo*" each topic was prefixed with the robot identifier. For instance, if odometry topic was "*odom_diffdrive*", it would be renamed as "*robot_1/odom_diffdrive*". This avoids interferences between robot's topics.

3.2 Mapping & Navigation

This part summarises the methods employed for mapping, navigating and path planning.

3.2.1 Mapping

Before path planning can be performed, a mean of mapping is required. Therefore, after an obstacle avoidance algorithm was developed (explained later on the project), a SLAM (Simultaneous Localization and Mapping) mapping technique was chosen for performing this required mapping.

This method uses sensor and odometry data to map and localise itself simultaneously. It is an occupancy grid mapping that uses probabilistic from the taken data to determine if there is or there is not an obstacle in a certain cell.

To do this, ROS uses a third-party package called "gmapping". Therefore, the only need to map the maze is to understand how SLAM works and to configure this package

to adapt it for each purpose. As the robot model is quite close to the “Turtlebot” model [3.8], the configuration of this was applied. This algorithm uses a highly efficient Rao-Blackwellized particle filter (filters are techniques used to solve the probabilistic localisation problem). It takes the movement of the robot and the most recent observations to reduce pose uncertainty. It returns a 2D Occupancy Grid Map (OGM) which sets each cell from 0 (no obstacle) to 100 (obstacle) depending on the probabilities that each cell has on being an obstacle. [3.9]

Once mapping is finished (Rviz is used to visualise it), it is saved into two files: a YAML and a PGM (Portable Grid Map) file. The first one holds metadata about the map and the second one is a “*. pgm*” image of it.

3.2.2 Navigation

Navigation was employed for two situations in the project. Firstly, it allows to save coordinate points of the headquarters and the light spots found. Secondly, it performs path planning and navigates back to the headquarter once a resource is found.

Localisation and navigation used a created package holding a configuration folder for the path planner, global and local cost maps and a launch folder for the AMCL (Adaptive Monte Carlo Localisation). This used the Husky navigation package by adapting it to the robot model and project purposes [3.10].

3.2.2.1 AMCL

Once the robot has a map, it needs to know its position on it (localisation). The AMCL package provided by ROS achieves this. It is a 2D probabilistic localisation that implements the adaptive Monte Carlo approach using a particle filter to track the robot.

Monte Carlo samples points from a probability distribution to approximate it (i.e. it uses a set of points to stand for the belief of being the robot in a certain pose). Each point is a possible (x, y, theta) position. When the robot moves, using action and perception updates, the localisation becomes more accurate and the points fall closer to each other reducing the number of particles once the uncertainty gets smaller. [3.10]

AMCL takes as inputs the laser, map, and transformations data. It is launched through the main navigation launcher and takes as arguments the map and footprint frames. AMCL launcher as well as the main launcher of mapping and navigation code are shown in Appendices [4,5]

3.2.2.2 Path Planner and Moving Robot with Goals

Once mapping and localisation are performed, the path planning with its movement controller needs to be set up in a separate launcher. [3.10]

This launcher uses the following configuration YAML files that implement the actual path planning:

- **Cost map Common:** It defines the parameters used by the planner like robot and map frames, map and laser topics and inflation radius. This last constant defines the amount of obstacle inflation to avoid collisions with obstacles (doing this allows the robot to be considered as a particle in the inflated map space which reduces complexity).

- **Global Planner:** ROS provides three global planners: “navfn” (grid-based that uses a navigation function to compute the path), global (more flexible than the last one) and carrot planner (tries to get as close as possible to the goal even if it is an obstacle). As the goals would never be obstacles, the best choice was the second one due to its flexibility. [3.11]

- **Local Planner:** ROS provides three types: base local planner (implements the Dynamic Window and Trajectory Rollout approaches to local control which is a controller that drives the mobile base in a plane using the cost maps), the “eband” local planner (elastic band method) and the “teb” local planner (timed elastic band for online trajectory optimization). The first one was used to implement the inflation radius system. [3.12]

- **Planner:** It sets robot’s movement characteristics when going to a goal and when creating the path. Things like maximum and minimum velocities and accelerations, goal tolerances, local tolerances and so on are defined on this file.

A problem found when using the path planner was that the launcher was not able to publish over the velocity topic as it was creating a different topic publisher. Consequently, a C++ supporting file was created. It subscribes to the path planner velocity topic, taking its data in and publishing it over the actual velocity robot topic.

To adapt path planning and localisation to a multi robot system, in the navigation launcher a “group” robot tag was. Moreover, a fake localisation node was added. It makes easier the simulation as it does not consider odometry data so precisely as it would do with the real world (otherwise, it consumes computer resources that overcharge the PC CPU).

The original plan for mapping and path planning with the S.R teams was to map with one robot and send that information over a topic while the second team member was exploring and path planning for getting resources over the constantly created map. This would also allow the first robot to save light spot position information and send it to the second robot.

However, after researching and asking Dr Mehmet (professor of Computer Science in University of Leeds specialised in AI), this option was not considered viable. The greatest problem found was that it was too abrupt for path planning clearly and would lead to many problems with the robot.

Another option was to created map files after certain time period by the first robot and then add them together. Nevertheless, the greatest issue with this was to localise the robots on these new created maps which would be impossible as they do not know the relative position of the previous map information.

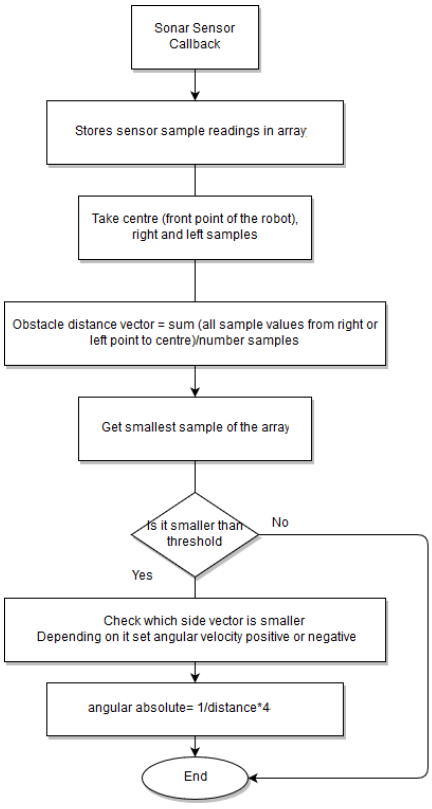
Finally, the problem was sorted out modifying project plan. Thus firstly, one robot maps all the environment. After an average period of 10 minutes, the map is detailed enough to save and both robots can start gathering the “resources”. Despite not being as efficient in big environments as the original plan, in a medium or small region, it is efficient and ensures stability and success.

3.3 Main Code for the Project

The main code has been hold in two very similar C++ files (this language was preferred to program them as the hardware interface was also written is this language) for each robot of each team. It performs the mapping, path planning and gathering algorithms (being this last one the most important and the subject to study). The other team holds the same code files but with the appropriate topic and frames naming changed as well as with a different resource gathering algorithm.

The file subscribes and publishes to different topics to process the sensor data. It subscribes to the sonar and light sensors. Moreover, it publishes to the velocity and chatter topics, being the last one where the robot communication is done and which also subscribes to (this topic interchanges between robots the position of the headquarter, the light spot points found by each robot and the actual position of each one). Each subscribed topic has a call-back function (something similar to interrupts in C++) that is executed with a frequency of 1kHz.

In one hand, sonar sensor call-back function is responsible of the obstacle avoidance algorithm. A method that robots can use without a map (in fact, used by the first robot that maps). The flowchart explaining this code is the following:



Note: from right/left samples, it adds up all the data values from their side to the centred ray and divides this value by the total number of samples taken.

This method allows the robot to sense not only what is to its sides, but also to know the average distance to each side. In the real-life world, these vectors are replaced by the two sonar single readings coming from the physical sensors.

In the other hand, light sensors call-backs are identical. They store the illuminance value caught by the camera into a variable to their later use.

Obstacle avoidance algorithm uses the sonar sensor callback for avoiding obstacles setting the angular velocity and if there is no obstacle within the threshold, it moves towards the area with most light gathering for light spots by doing so. Potential fields inspired this algorithm. The robot is repelled by obstacles and attracted by a goal that is in this case the light intensity. Using more angular velocity that the one needed to avoid an obstacle (four times higher) helps the robot to get out of local minima and avoid looping in the maze as it is turned more than what it should be. This changes its trajectory with a certain random value. This decision of setting it four times larger came from testing and trying to get the minimum value for getting out of loops (therefore, allowing to explore all the maze).

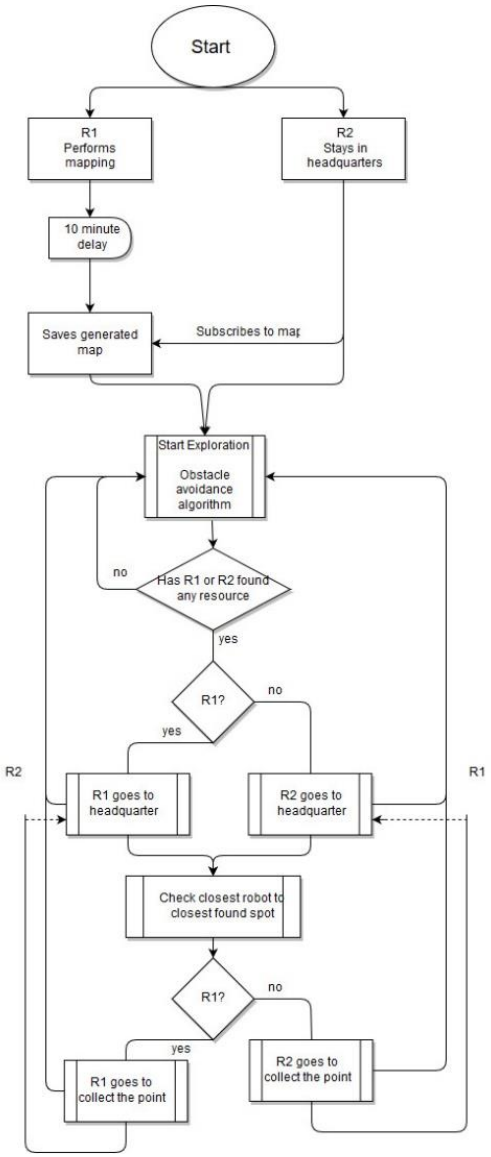
For the rest of the code, the resource gathering algorithms resigned and tested are used. For moving to a certain coordinate, a method of the ROS class “move base” is used which sets the goal to that point and performs path planning. The tested algorithms will be explained now and the results of testing them showed in “Chapter 5: Results & Key Findings”.

3.3.1 Search Swarm Robotics Algorithms

This is a brief explanation and justification of the methods designed to coordinate the two groups of robots for a posterior analysis and comparison shown is the results chapter. Although the algorithms are implemented in the main code file, this section only reviews its general functionality in a flowchart. The actual code implementation can be seen in the appendices.

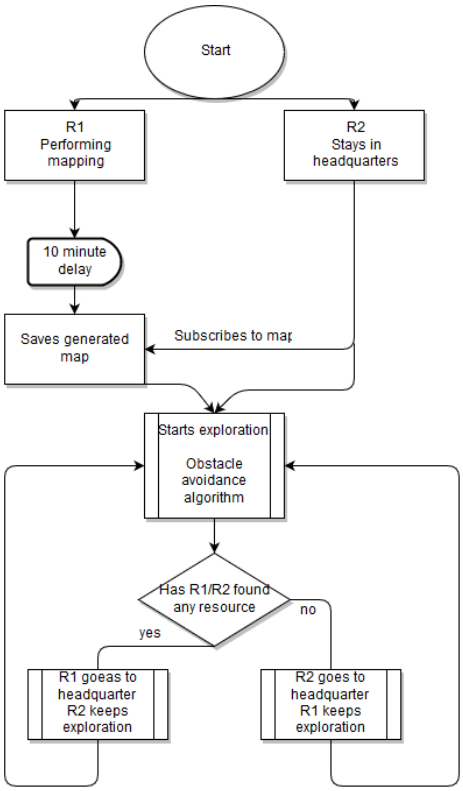
3.3.1.1 Algorithm 1

R1 and R2 stands for the first and second robot in each team. This algorithm performs two main activities using coordinated swarm robotics. The first activity is the exploration looking forward for finding new light spots. The second one is to collect the “resource” from the already found spots. Therefore, the system is not risking neither to get close to discover new relevant resources neither to spend too much time on exploring the environment. These two activities are performed parallelly and trying to optimise the time range and the battery charge for a good efficiency. The code unfinished for this algorithm for the first robot is on Appendices [7].



3.3.1.2 Algorithm 2

As in the previous case R1 refers to the first robot of the team and R2 to the second one. This algorithm was created to compare to the previous one and therefore, test whether one that takes advantage of coordination between robots, can get better results than the other team that makes each robot to explore and seek for resources without information exchange.


Start
R1
Performing
mapping
R2
Stays in
headquarters
10 minute
delay
Saves generated
map
Subscribes to map
Starts exploration
Obstacle
avoidance
algorithm
Has R1/R2 found
any resource
no
yes
R1 goeas to
headquarter
R2 keeps
exploration
R2 goes to
headquarter
R1 keeps
exploration

Chapter 4: Hardware Design Middleware

This chapter covers the hardware robot design, its development and the the code that interfaces the components.

4.1 Specifications

During the first stage of the project, a S.R standard model was contemplated to be created under a concrete number of specifications. These are dependent on the principles of S.R as well as in the time constraint and budget constraint limitations. These specifications were:

- Small-sized and light robots.
- All of them must be based on the same model (as they belong to a S.R system. they need to be homogeneous for avoiding privileges between the teams and between individuals).
- As simple as possible. The aim of the project is to give a very simple robot the ability to expand its constrained abilities by collaborating with the other team member.
- Non-holonomic robots to keep simplicity in trajectory calculations and expectations. Moreover, they are required to be powered by two motors and to have one or two support caster wheels to ensure stability.
- Capable of sensing their surroundings in a straightforward way for making lighter the data transfer.
- Capable of communicating wireless via Wi-Fi between themselves and the PC.
- As cheap as possible.

These requirements lead to a deep component seeking process that tried to meet these parameters.

4.2 Components selection process

This section shows the components considered for the design and justifies the final choice depending on the above specifications, especially the ones related to simplicity, size, weight and cost.

4.2.1 Microcontroller and Wireless Communications

The robots have two main microcontrollers (MCU). The one that interfaces with the peripheral components (sensors, motors, battery and encoders) and the Wi-Fi module connected via serial communication to the first one (it sends and receive the relevant sensor, odometry and velocity data from the rest of the robots and PC wireless).

Starting with the first one, it was suggested by the supervisor that ATTiny was the best choice for small MCUs (it is the ones used by Arduinos). Due to the weight and size requirements, it was decided to use directly the MCU with no board controller like Arduino or Mbed. This reduces costs, size and weight although it increments the difficulty of programming and interfacing.

These were the considered options to choose:

Name	Price	Brief Description
ATtiny2313	1.86£	MCU with 18 pins and 10-bit ADC that has an operating voltage of 2.7-5 V and a memory size of 128 B EEPROM sRam and 2 kb Flash.
ATmega168-20P	3£	28 pins (2 8-bit ADC and 6 PWM), running at 20 MHz at 2.7-5 V and with a memory size of 512 B EEPROM. Serial and parallel connecting with USART, SPI and WDT.
ATMEGA8U2-AU	3£	22 pins MCU which best feature is its very small size, it has communication through SPI and USART.

Table 4.1: MCU options

For purpose of the project ATMEGA168 was the one with the smallest size and lowest price that was giving more features apart from being one of the most used in Arduino boards along with ATMEGA328 so there would be enough documentation on it.

To choose the wireless module, some parameters needed to be prioritised. These were the communication range to be neither too short neither too long (to avoid interferences), communication area, message length and its propagation time. [4.1]

Name	Price	Brief Description
ESP-12E	4.48 £	OV at 3.3V, frequency of 2.5GHz and memory size of 4Mb Flash. It allows communication over the following protocols: I2C, UART, HSPI, I2S, IR remote control and TCP/IP. It includes ports GPIO and PWD.
ESP8266 SparkFun	12£	Wi-Fi, direct P2P, TCP/IP, it is a supportive MCU for the module (20 pins & on-board charger/PS).

ESP8266 Wi-Fi module	5.23£	1Mb Flash, wake up and transmit packets in less than 2ms TCP/IP, UART, SPI integrated PLLs, small, 8 pins, integrated regulators and power management.
Wi-Fi and Bluetooth module ESP32 with external antenna	6£	Bluetooth and Wi-Fi module, small and practical. Might need an antenna.

Table 4.2: Wi-Fi module options

Considering that ESP-12E had a AT firmware already installed and that it provides its own antenna, it was seemed to be the best option. The only inconvenient was that it only provides Wi-Fi communication and not Bluetooth as well.

4.2.2 Sensors

To sense objects in the environment, the robots need to use sensorial perception. Diverse types of sensors were considered:

Laser sensors/ light phototransistors: Although widely used, these have the inconvenience of depending on ambient light and light changes as well as having their data being altered depending on the material or colour of the object. These were the main reasons for not choosing them despite considering possible options like a proximity sensor with a PCM-filter like [4.2].

Bumper sensor (Atmel AT42QT1070): the main reason this was not chosen was that it only allows to perform any of the bug algorithms for mapping and path planning which are quite limited in performance. Therefore, despite being a possible feature to add to the robotic sensorial system, to keep it cheap and simple they were not used. [4.2]

The final choice was to use the general and widely used ultrasonic HC-SR04 sensors. They are the ones that best fit to the aim of the project as they work very well over plane and close surfaces (a maze is their ideal environment). Moreover, they are not light dependent. Their greatest inconvenience is that they only emit one data point per sensor and not a range of them and they can get a great amount of noise if not in an ideal environment.

4.2.3 Power & Locomotion

The main of locomotive system are the motors, encoders and wheels while for power system a single battery is required.

The motors should be very light, small and as powerful as possible. Therefore, the possibilities turned around micro gear motors.

Name	Price	Dimensions (diameter, length and weight)	Power/Torque/Voltage/Current
GM15-solarbotics	10.46£	6 mm D 20.1 mm l 1.3 g	3V nominal operation 920 rpm – 100 mA 35.3 gcm
Sparkfun-Standard Gearmotor	18.68£	6 mm D 20 mm l 128 g	3-12 V nominal operation 10rpm at 12V 300:1 G. R
Sparkfun-Micro Gearmotor	9.72£	6 mm D 10 mm l 17g	6-12 V nominal operation 45-90rpm 298:1 G. R
Hobby Gearmotor (DG01D)	3£	6 mm D 10x80 mm l	4.5 V nominal operation 140rpm 48:1 G. R
FA-GM6-3V-25 Micro Motor Mini Motor [14]	2.24£	6 mm D 20 mm l 1.2 g	3 V DC nominal operation 1200 rpm 20 gcm 100 mA

Table 4.3: Micro Gear Motors

Pololu micro motors were selected considering the above parameters. Encoders and wheels adapted for the size of these motors were also available which was a determining feature for their choice.

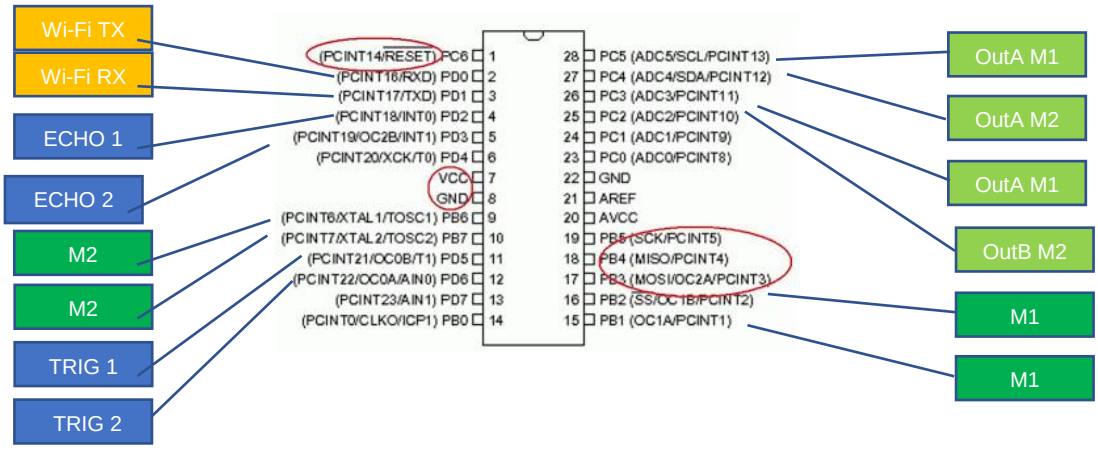
Finally, the batteries needed to be small and rechargeable. The best choice was Lithium Ion Batteries which have a wide range of charge life time to select and are one of the smallest and cheapest ones in the market.

4.3 Robot Final Design Datasheet

This section discusses the final design with all the parameters of the robot model. Its aim is to show the methods used and the design applied and to prove its viability.

One of the biggest challenges and aims was giving a robot that can be used in the future by other students, teachers, professionals or researches, so that time can be saved as well as money and resources. Moreover, creating a flexible model also allows to easily replace faulty components without the need of changing or breaking down the circuit or the robot. Its circuit and assembly is made in a way that it can be fast be unassembled for its reparation and allowing as well to look into problems of the robot in a clear way thanks to a tidy design.

ATMEGA168 connections diagram is the following:



The pins surrounded by red circles are the ones used for connecting to the programmer.

In the next sub-sections each component its main characteristics and code will be explained.

4.3.1 Sonar Sensors

HC-SR04

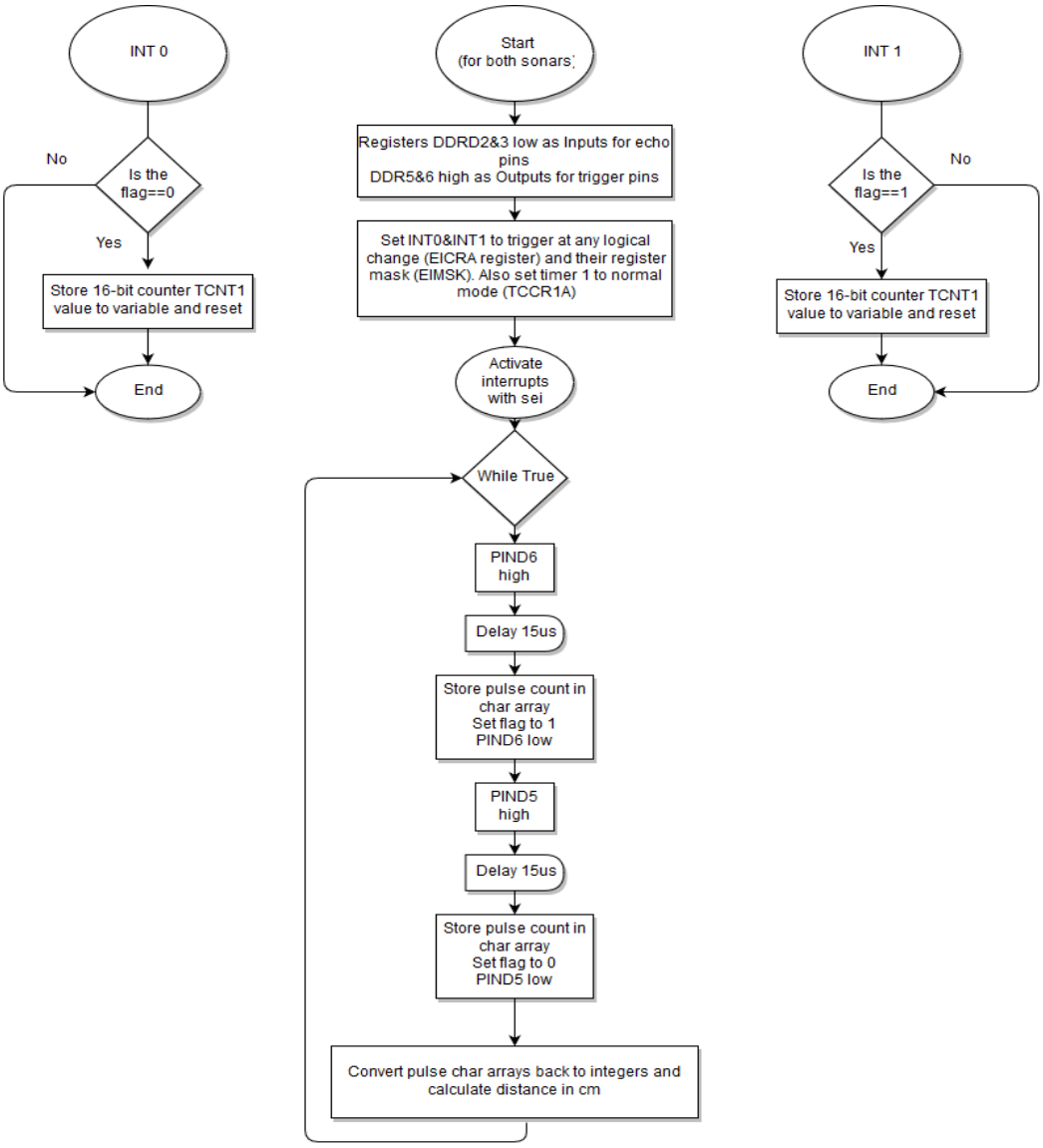
Type	Ultrasonic Sensor
Pins	Vcc (5V). It has been tested that it can also work with the circuit voltage at 3.8V.
	Trigger. It sends a high pulse for more than 10us. Then, waits for 50us for sending the ultrasound signal again.
	Echo. Set as an input. It receives the sonar data sent from the trigger back.
	Ground.
Range	2cm-4m. Enough distance for the maze.
Angle	15º
Functionality	Firstly, by outputting the trigger pin on for a period over 10us, an ultrasonic wave created by 8 acoustic bursts is sent at 40kHz. Once it hits an object, the wave is reflected. After triggering, for 50us the line is set low and the timer starts counting. When echo receives the signal back, the timer stops. The distance is calculated considering the time it took the signal. Considering that in normal air the velocity of sound is 340m/s, the distance is: $Distance(cm) = Pulse\ width\ in\ \mu s/58$
Pros	In an environment with plane surfaces like a maze, its functionality is very good. It has a good range and it does not need many support components to get to work. Light does not affect it.
Cons	A single data point is generated leading to 1D representation of the space and not 2D like it could be with a multi sample laser sensor. Moreover, very little accuracy is obtained over non-planar surfaces. It also has a short angle and the quality of the data depends widely on how the sensors are placed in the robot.

Table 4.4: Sonar Sensor Component [4.3]

Code

The interface with these sonar sensors and with the rest of the components of the robot is done through Atmel Studio, an environment for programming MCUs of ATTiny and in a lower computing level than Arduino IDE.

The following flowchart explains sensors code interface:



INT0 and INT1 are external MCU interrupts. One important thing to clarify is that both use the same 16-bit timer so they are coordinated by a flag to avoid data overlapping or interfering with each other's counter. [4.4]

4.3.2 Motors

Pimoroni COM0800 Micro Metal Gearmotor 50:1

Type	Gear Motor 50:1
Speed (6V)	420rpm 10%. From several tests performed, they start moving at 3.5V and from 5V the difference in power cannot be appreciated. At 3.8V (circuit voltage) their working velocity is 0.6m/s.
Stall torque (6V)	0.5kgcm. From several tests it has been confirmed that enough torque is provided for moving the robot easily and with no overheating in the motors.
Stall current (6V)	770mA
Shaft Diameter	3mm D-shape, 9mm long.
Pros	Light, small-sized, enough torque provided.
Cons	The need a power control system like an H-Bridge. If a wall is heated by the robot or if it gets stuck motors experiment overheating.

Table 4.5: Micro Motor Component

Code

Motors are activated by setting their DDRB registers high as outputs and then being controlled through the code depending on the closeness of the obstacles.

4.3.3 H-Bridge

VMA409 - LM298N Dual Bridge

Type	Dual Bridge DC Stepper Controller Board.
Pins Used for the Model	Vcc (5V) although it works at the circuit operating voltage of 3.8V.
	Control Pins. Four of them used, two for each motor each one for clockwise or anti-clockwise rotation. By inputting these pins high or low the motor is on determining its direction of movement.
	Motor pins. Four pins connected to the motor inputs. Two in each side of it.
	Ground.
Dimensions	69 x 56 x 36 mm. One of the largest components. It is seated alone in the lower plate of the robot structure.
Functionality	The main advantage of this board is to give a good and trustful motor control successfully and without putting in risk the microcontroller. It has an H-Bridge circuit in the board which allows to control motor´s direction successfully using p and n-channel transistors and protects the MCU against back electromotive current created after the motor changes state. Moreover, it gives a suitable current and voltage to the motors.
Pros	Protects the motors and MCU, brings a good controlling system and a fast and easy way of assembly.
Cons	It is not on the main circuit board which adds more cables around and adds more weight to the robot.

Table 4.6: H-Bridge Component [4.5]

4.3.4 Encoders

Pololu 2591 Optical Encoders for Micro Metal Gearmotors

Type	Quadrature Optical Encoders
Pins	Vcc (3.3V). A potential divider reduces the nominal voltage from 3.8 to 3.3V suitable for the encoders.
	M1, M2. Pins not used. If connected to motors input, they can be controller through these pins. For the purpose of the project it was better to provide an H-Bridge control system instead. Thus, the encoders can also be replaced or unattached to the motor easily as they are not welded to it.
	OUT A, OUT B. Output signal from each optical encoder.
	Ground.
Encoder wheels	As position and velocity do not need to be extremely precise (in the range of ±3 cm), a three-tooth encoder wheel was used instead of the 5-tooth one. The fabricant recommends using this one as outputs tend to improve with this wheel.
Resolution	12 ticks when using 3-tooth wheels.
Functionality	The odometry of the system is provided by these components. They return a high or low voltage from two optical sensors. These sensors send a light ray and if the wheel is hit by it, returns the signal and the voltage would be high. If both encoder signals are high it means they are synchronised and therefore, the wheel is going clock-wise. By determining the number of times, the signal goes high, the distance and velocity can be calculated (information below).
Relevant calculations	The calculations and the parameters needed to perform the adequate encoder calculations were the following ones: Motor gear ratio: 50:1

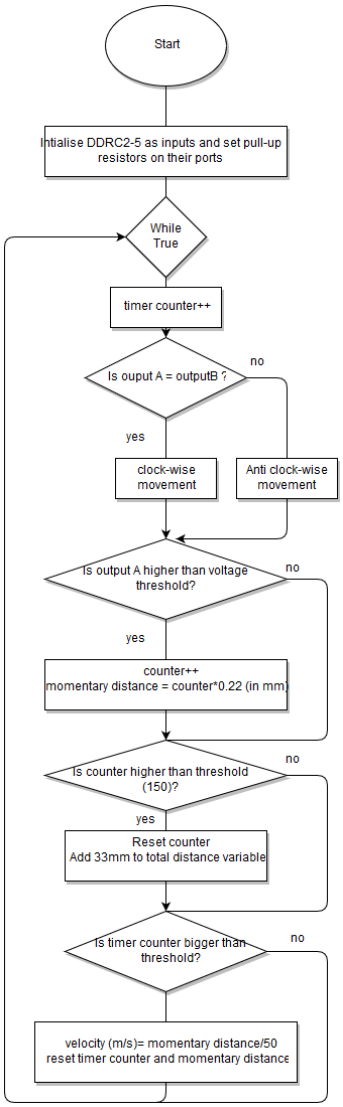
	Motor velocity (6V): 420rpm=7rps Encoders 12 ticks per revolution. One-wheel revolution: $gear\ ratio\ x\ number\ ticks = 50x12 = 600\ ticks\ per\ revolution$ One-wheel revolution: $2\pi r = 42\pi = 132mm = 600ticks$ Distance per tick: $\frac{132}{600} = 0.22mm$
Pros	Small-sized, light and fit with the motors (they were designed for that model). High resolution.
Cons	Light dependent which means that they should be placed where ambient light changes are. Pins difficult to sold due to their small size (2mm pitch). Works at 3.3V. Their signal is sinusoidal instead of digital (more complex signal processing leading to data noise).

Table 4.7: Encoders Component [4.6]

Code

Encoders are set with their DDRC registers as analogue inputs despite throwing back a sinusoidal and not digital sign. This might reduce accuracy but for the purpose of the project is fair enough. They are interfaced using analogue inputs with pull-up internal resistor activated.

The code that interfaces them can be summarised on this flowchart:



Output A and Output B refer to the signal coming from the first and second optical encoder placed on the same motor. Each robot will have two sets of optical encoders which means that it will have a total of four optical encoders. [4.7]

4.3.5 Wi-Fi Module

Wi-Fi ESP8266
ESP-12E

Type	Espressif ESP8266 Wi-Fi Microcontroller.
Used Pins	Vcc (3.3V). It needs to receive the nominal voltage from the potential divider.
	TX, RX. Serial transmitter and receiver. The data from ATMEGA168 is interfaced through these serial pins for being able to make the MCU send and receive data wireless. UART serial communication.
	Enable. Always connected high with a 10kΩ pull-up resistor.
	Reset. Connected high to a 10kΩ pull-up resistor in normal mode. In reset mode, it needs to be connected to negative and then, back to positive.
	IO0. If high using a 10kΩ pull-up, it is in UART serial interface mode allowing to reprogram the firmware of the module via serial communication. If low with a pull-down resistor, it sets the module to Flash the Bootloader.
	IO2. Set high with a pull-up resistor of 10kΩ for selecting UART of bootloader mode.
	IO15. Set low with a pull-down resistor of 10kΩ for selecting UART of bootloader mode.
	Ground.
Security	WPA2
Memory	4 MB external SPI flash, 36kB of SRAM. The project software occupies around 50% of the total memory due to the amount of string data conversions.
Range	90 m. [4.8]
Functionality	The module receives serial data in form of chars from ATMEGA168 and passes it in

	form of strings to the web host created. The website shows odometry and sonar sensor data and can send velocity commands to the module. This, converts this data back to chars and they are converted in the ATMEGA168 back to integers to be used by motors.
Pros	Full Wi-Fi front-end both as client and access point with AT firmware installed by default. Small-size and light, provides extra GPIO apart from Wi-Fi interfacing. An antenna of 90m range is included. Can be set to act as a server.
Cons	Do not work with Atmel. Therefore, for creating the firmware to send data wireless Arduino IDE has been used. Moreover, it does not have a website interface where to receive and send information. An external host needs to be created through AT commands or firmware with a website and a database that supports and stores this data.

Table 4.8: Wi-Fi module Component [4.9]

Code

The module uses Arduino IDE for being programmed and for communicating serially and wireless to the PC. It is connected serially to the ATMEGA168 from where it gets char arrays of odometry and sensor data and converts them into string arrays for sending to the website created for this project. It sends also char arrays previously converted to the MCU with data about motor's output for velocity and trajectory control from the website application.

The SSID, password and web host are first stored into variables for connecting wireless. Then, serial communication starts at a baud rate of 115200Hz and requests the Wi-Fi connection. Afterwards, it is set as a client to this host server and a URL variable is used to hold all the desired parameters to send or to receive. [4.10]

4.3.6 Microcontroller

ATMEGA168-20PU

Type	Low Power 8-Bit MegaAVR MCU.
Used Pins	Vcc (1.8-5.5V).
	TX, RX. Serial communication with the Wi-Fi module.
	RESET, MOSI, MISO, SCLK, VCC & GROUND. These 6 pins are required for connexion with the AVRSPII programmer. First one is the rest line, second and third the master out slave in and master in slave out pins, fourth the internal clock and finally the power and earth pins.
	PD2-3. Inputs set as interrupts INT0 and INT1 for the sonar sensors echo receiving signal.
	PD5-6 Digital outputs for the trigger of the sonar sensors.
	PB6-7, PB1-2 Digital outputs for the motors.
	PC1-0 Inputs for the light sensor signals.
	PC5-2 inputs for the two odometry signals from each motor.
	Ground.
Timers	TCNT1 16-bit timer used for sonar sensors and two 8-bit timer.
ADC	6 10-bit pins.
Internal clock	128kHz.
Functionality	MCU that interfaces with sensors, odometry, locomotion, power system and Wi-Fi module. Receives and sends data to the rest of the robots as well as to the PC and an obstacle avoidance algorithm has installed on it.

Pros	Small-size, light and low power consumption. It satisfies all the expected features for the project: ADC, PWM, UART serial communication, timers, high number of GPIO and a wide range of operating voltages.
Cons	A programmer provided from the University needs to be used to download the code on it or to debug it. Atmel is the software needed which is in a slightly lower computing level than Arduino. Moreover, GPIO cautions need to be taking for working with it opposite to an already created board. For the project purpose a created circuit Veroboard had to be created to satisfy these requirements. Moreover, it has only one 16-bit timer and only one 8-bit timer that can be used as TNCT0 is exclusive to MCU timing purposes.

Table 4.9: MCU Component [4.11]

Code

The MCU communicates to the Wi-Fi module using serial communication following the tutorials of []. It also has the code for avoiding obstacles in case the website is not used for controlling the robot based on the sensors data. If an obstacle is not within the minimum distance the robot moves straight. Otherwise, it moves to left or right for avoiding it and in case it is too close it stops and turns in its own place using only angular velocity. All the code for the Atmel programming an component interfaced is shown in Appendix [8].

4.3.7 Battery

Polymer Lithium Ion Battery (3.7V, 1Ah)

Type	Li-Po Battery
Pins	Vcc (3.7V). From different tests it has been determined that this voltage varies between 3.6-3.85V.
	Ground.
Dimensions	34.5x6.2x52mm, 100mm JST cable
Capacity	1000mAh.
Functionality	Rechargeable batteries.
Pros	Small sized, very light and rechargeable.
Cons	Can explode if not been manipulated properly and require a special charger that was provided by the University.

Table 4.10: Battery Component [4.12]

4.3.8 Supportive Components

4.3.8.1 Caster Wheels

Ball Caster Metal - 3/8" & Iron Ball Transfer Bearing Unit Ball Casters Wheel

Type	Cater Wheels.
Dimensions First	3/8"diameter and with two spacers.
Dimensions Second	4mm hole with 15mm diameters of the metal ball.
Functionality	Supportive wheels that provide stability to the robot.

Pros	Small sized, 3 degrees of freedom.
Cons	The second ones due to some issues had to be iron ones and thus, quite heavy.

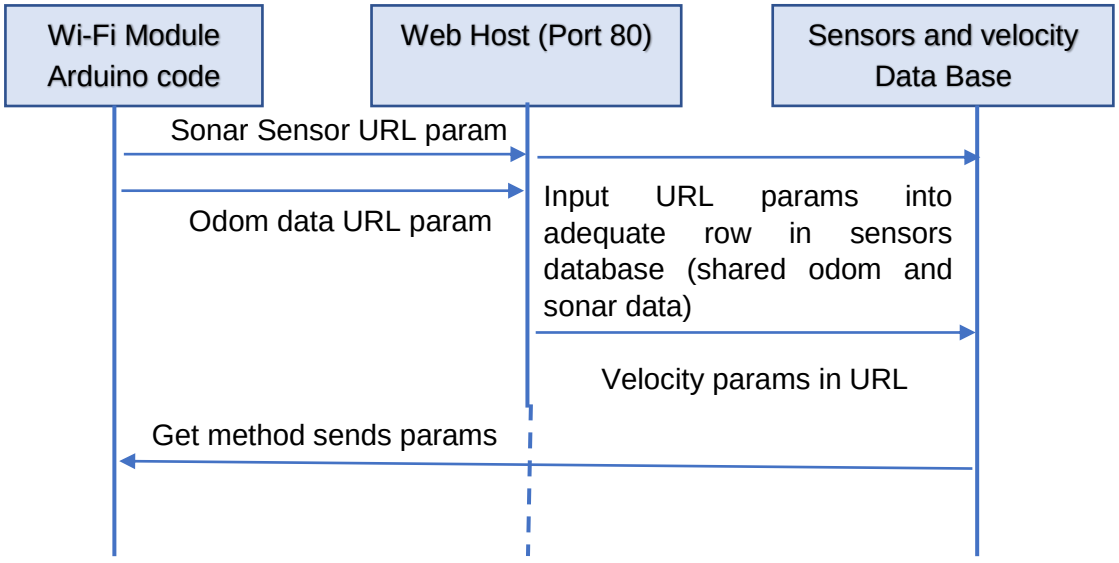
Table 4.11: Caster Wheel Component

4.4 Website Supportive Software & Communications System

This section covers the website interface created with the aim of acting as the server between the four robots, displaying the relevant sensorial data and being able to wireless control the movement of the robots. The system was inspired by [4.10] tutorials and change it to suit the project purpose.

A database using MySQL (accessing through MyPHPAdmin) and the web host were created using "www.000webhost.com" which allows to create them for free for not complex purpose projects allowing to upload back and front-end files.

The structure of the client (Wi-Fi module) server (website on PC) was:



Front-end basically sets the HTML5 and CSS3 visual website information apart from JavaScript (JS) scripts inserted on it (all these languages are applied over the same "*index.html"* file as the website was not large). These JS takes the URL as input and the database (DB) information using JSON and connects to the back-end.

On the back-end side, it has seven files for receiving data and four for sending. Starting with the first case, "*config.php*" defines the database, Wi-Fi password, username and the server, in this case the localhost of the PC.

The connection to the databases is performed by a class in "*db_connect.php*" which taking the parameters from previous files uses MySQL for opening and closing the connection and for selecting the desired database the client wants to access.

The rest of the files are for inserting one request into the database, for deleting one row from it, for updating it and for reading all or a single row. These are the basic HTTP actions (UPDATE, INSERT and DELETE). An example code of the insert file is shown in the Appendix [9].

The procedure in all of them is very similar: reads the URL looking for the parameters, connects to the DB, makes a MySQL query and checks if the request went well.

The back-end for sending the velocity control is very similar to the one for receiving data. The difference is that in this case, it only needs a read file and an update file the values are fixed and they only need to switch between each other.

This was found to be the better choice to configure the multi robot communication. Other workable option was to set one of the modules as the server and pass from it a HTML5 code that it would have in its firmware. Due to the memory space this would occupy this was not chosen. Moreover, other websites for controlling wireless the ESP module were checked but all of them were for controlling the basics (LED control) or for sending data collected by the module itself and not taking it from the serial communication.

One of the most challenging parts was the data conversion. ATMEGA168 does not have much memory due to its size so arrays of chars instead of strings were used for saving space. ESP was converting them to strings and vice versa for being able to send them with the URL.

Chapter 5: Results & Key Findings

5.1 Simulation

5.1.1 Problems Found

This part analyses the problems found during the project in this field, the tasks left to do along with the reason of this and a strategy for avoiding it in future work.

The number of tasks finished in this part of the project were around a 90% of the total expected final work on it. The main reasons why these last parts were not able to be finished were:

- **Time constraint:** To develop a hole multi robot system for ROS and Gazebo from scratch and from a new robot model lowered down the rest of the project in this area. Moreover, path planning and mapping took also a lot of time as different systems had to be studied and considered as well as their implementation. However, the most time-consuming task was to adapt the robot model to be able to launch four independent robots moving with their own frames and topics.

- **Load of work:** To learn several new programming languages (URDF, SDF, YAML, bash and launch files and so on) and to learn ROS from scratch and at a high level [5.1] lead to a load of work at the robot model creation stage.

The tasks left to do from this part of the project were:

- Test algorithm 1 and 2 to get a heatmap of the light points registered in the maze and the amount of them got in 10 minutes.

5.1.2 Experimentation Results

This block explores the main tests done over this part analysing them, comments over the expected outputs from the tests that could not be done justifying them.

Firstly, in terms of obstacle avoidance algorithm, the robot was implementing it properly most of the times. However, in positions like this one:

PHOTO

The robot was struggling to get the minimum separation which was leading to occasionally hit the wall and get stuck with one of the wheels on it. The reason why this was happening was because the total distance vector as the obstacle was only being detected by a few points was in average high. As a result, the turning velocity for avoiding the obstacle was not fast enough. A way of overcoming this could be to make this velocity inversely proportional to the closest distance single point to the wall and not the average vector.

The multi robot system was spawning them adequately after testing their topics, moving them independently and looking at the general frame tree.

Mapping was done after several tests to check the amount of time taken to get a relatively complete map. These were the performed examinations:

Map Test	Time taken to create a complete map (minutes)
1	8:55
2	10:22
3	9:30
4	7:55
5	11:03

Table 5.1: Map Testing

Therefore, a minimum time of around 10 minutes was considered to be a good assumption to make sure the map was done and pass to the next part when automatization.

Finally, the last part was the one that due to the reasons explained above could not be performed. The code for the four individual robots was close to be finished although issue at the last days of work related to sending and receiving spot lights data points lead to not finish them.

However, they had the ability to perform most of the algorithms 1 and 2 and they were given code for plotting the points found in a text file. It will be tried, before physical demonstration of this work, to finish the code for the final demonstration and in affirmative case the results that will be explained before will be send to the supervisor.

The tests and results to be performed in future work are:

- Efficiency graph: number of spots found in ten minutes per team.
- Heat map: map showing the points with the highest number of spots found.
- Comparison: comparison in the two above tests from algorithm one to two.

The expected outcome and the hypothesis for future work is that the first algorithm would be quite more efficient than the second one with a less distributed heatmap as the robots go to the already found light spots. This would confirm that S.R are better than many individual robots.

5.2 Real Robots

5.2.1 Problems Found

This section looks at the work done in the hardware side, the problems found, the reasons that let to not finish all the expected parts and the strategy to take for avoiding it to happen in future works.

Although 80% of the expected work in this area was finished, there were a couple of tasks not completed. The reasons were the following ones:

- **Time constraint:** The main reason for staying behind the expected time was related to the first stage of the research. Too much time was spent in this part of component selection which lead to delays in the rest of the project.

- **Load of work:** The amount of work due to soldering and mounting four robots and specially the one coming on directly interfacing the ATMEGA168 could have been reduced leaving more space to the unfinished tasks. Instead of using the MCU directly, a better option would have been to use Arduino which already comes with libraries for interfacing with components and with a higher lever programming that avoids interfacing so many technical details like registers, timers, clocks and so on.

- **Compatibility issues:** Simulation software has a high computational cost and a high complexity that neither the MCU neither the ESP 12-E had. This fact was underestimated. To perform the simulation into real models would need to be carried through a central PC. A commands system through the website would be required where the ROS nodes would send and receive data from real robots caught by PC wireless connection to them.

The parts left to do for the Hardware were:

- Finish adding a light sensing system.
- Connecting the simulation software to each robot wireless using a series of commands for communicating with ROS.
- Testing website communication with all the four robots simultaneously and not individually.

Moreover, during the project, there were several unexpected problems that led to modify the original plan or design. These were:

- Use of two sonar sensors instead of three. Due to practical reasons, it was determined that sensing could be done with a minimum of two sensors and not three.
- Use of four total wheels instead of three. It was found from experimentation that for keeping stability the minimum number of wheels were four.
- Use of board manufactured H-Bridge instead of self-made ones. Due to difficulties and time-consuming factors, this was decided to be the fastest and safest choice.

5.2.2 Budget

This section englobes the main experimentation made over the built robots with an analysis of its performance and the possibilities the robot would have for future works. Moreover, a table with the total price is shown below along with the closing price per robot:

Component	Quantity	Price per unit (£)	Total price (£)
ATMEGA168-20PU	4	2.30	9.20
Wi-Fi Module - ESP-12E	5	3.53	17.65
Polymer Lithium Ion Battery (3.7V, 1Ah)	5	5.46	27.30
Micro Metal Gearmotor 50:1	8	4.62	36.96
HC-SR504 Ultrasonic Ranging Module	12	1.90	22.80
Optical Encoder Micro Metal Gearmotors	4	7.78	31.12
Pololu 1090 Wheels 42x19mm	4	5.20	20.80
USB to TTL Serial Converter	1	15.30	15.30
Dual Bridge Stepper Motor Controller	4	7.95	31.80
Crystal 16MHz Oscillator	8	0.252	2.016
Total			214.946
Price per robot			45.26

Table 5.2: Project Budget

The TTL converter was required for serial communication to the PC from ATMEGA168 and ESP 12-E as University did not have any. The oscillators were required for experimentation with external clock and oscillators on the ATMEGA168 but not decided to be implemented in the final design.

This price per robot could be decreased if the H-Bridges were manufactured instead of ordered and if the optical encoders were also self-manufactured by ordering optical mini sensors in a PCB circuit. The smallest price therefore could be reduced up to £29.73.

5.2.3 Experimentation Results

The experiments realised in this hardware section were for testing the performance of the code and circuit controlling each component and to test the whole system at the end.

Sonar sensors, after testing on the final model were found to be precise in the desired maze environment. However, they were only providing one data point which was leading sometimes to problems when the robot was turning or in a corner as it did not

have clear the precise distance to the wall. The reason is that the moving of the robot itself was altering this outcome. A better possible software would be to take several data points over time and to create a distance vector like it was done in the simulation.

Odometry was working although not extremely accurate as the input voltages were varying rapidly. The reason is that optical encoders are very affected by light changes and these in particular were using sinusoidal signals instead of digital ones. A better approach to this issue would have been to design a component for suiting them and protecting from ambient light and to take them through an appropriate ADC pin instead of an analogue one.

Wi-Fi module was working correctly connecting to the website adequately and to the serial communications of the ATMEGA168 as well. Data transfer could have been improved between the MCU and the module sending uint8 (integer data stored in 8 bits) instead of arrays with chars and strings. If more information would have need to be send this could be a problem.

The general performance of the robot avoiding obstacles is good although it “doubts” too much whether there is an obstacle or not leading to abrupt turns.

5.2.4 Conclusion

A valid but still improvable robot model has been built along with another three identical robots. All of them avoids obstacles. It has also been demonstrated that one of them can connect to the PC website despite not having tested neither finished the code that allows all of them to get connected. Moreover, components interface works despite having some points to improve.

The task left to do in this part of the project was to develop the final communication system to the ROS code used in simulation. This code would communicate to the created database, read sensor and odometry data, input it in the system and send to the robot through wireless communication the wireless velocity messages.

Conclusion

Possible extensions

As explained in the chapter 5, there were still some things left to do and, moreover, other extensions that can progress on the direction this project has started.

To sum up, below is a list with the possible and interesting extensions and work that can be done in future plans to keep with the aim of this project:

- Finish, test and develop more algorithms for refining the coordination and cooperation when gathering for resources in the maze.
- Add a camera or a light sensor to the robot models so that they can perceive changes in their environment. Moreover, to use a bumper to move backwards when an obstacle is accidentally hit would be also a good idea.
- Create a communication system based on commands between ROS programming and the actual robots that can only perform short calculations with their MCUs. The created web would be the middleware between these two sides of the S.R system.
- Test the simulation with more than two robots per team to observe the behaviour of the groups.
- Add to the resource gathering algorithm a way of stealing points (resources) from the other team to increate competence through conflict and implement some kind of Min-Max AI algorithm for predicting opposite team strategy.

Applications of the Project

Considering the results, the parts left and the total work realised, this project in terms of both software and hardware could act as a template or a guideline to develop S.R systems. It could also be used for next researches on this topic. Moreover, the methods suggested, if refined a bit more and tested, could be employed for resource gathering, rescue missions or life gathering in the space as well as general environment exploration.

Conclusions

This has been a challenging project involving a learning process in many engineering areas simultaneously and with many milestones. A suitable S.R system has been developed both in hardware and software with its discussed methodology justified for the final purpose of the work. Despite not finish connecting the simulation with the real world and despite not finishing testing the two optimising algorithms for resource gathering in S.R, the path to follow for finishing these last steps has been given.

To sum up, a robust simulation with its multi robot system using ROS has been developed as well as the design of S.R algorithms for exploration and gathering tasks. Moreover, a mapping and navigation system has also been configured to be used in S.R and a real robot hardware model following the principles of S.R has been designed and built along with a website to interface wireless to the robots. Finally, a roadmap to the next steps on developing fully the S.R system have been given.

References & Bibliography

[1.1] M. Boly, A. Seth, M. Wilke, P. Ingmundson, B. Baars, S. Laureys, D. Edelman and N. Tsuchiya, "Consciousness in humans and non-human animals: recent advances and future directions", 2018.

[1.2] *Hamlyn.doc.ic.ac.uk*, 2018. [Online]. Available: http://hamlyn.doc.ic.ac.uk/uk-ras/sites/default/files/UK_RAS_WP_SR25yr_web.pdf.

[2.1] Available:https://www.sciencedirect.com/science/article/pii/S221491471300024X

[2.2] Available:https://ww.researchgate.net/figure/The-principle-of-swarmrobotics_fig2_260037606

[2.3] J. R. Beckers, "From local actions to global tasks: Stigmergy and collective robotics", *Citeseerx.ist.psu.edu*, 2018. [Online]. Available:http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.568.

[2.4] "The emergence of artificial culture in robot societies", *Brl.ac.uk*, 2018. [Online]. Available:http://www.brl.ac.uk/research/researchthemes/swarmrobotics/artificialculturеinrobots.aspx.

[2.5] "Nodes - ROS Wiki", *Wiki.ros.org*, 2018. [Online]. Available:http://wiki.ros.org/Nodes.

[2.6] "Topics - ROS Wiki", *Wiki.ros.org*, 2018. [Online]. Available:http://wiki.ros.org/Topics.

[2.7] "Messages - ROS Wiki", *Wiki.ros.org*, 2018. [Online]. Available:http://wiki.ros.org/Messages.

[2.8] “Gazebo : Tutorial : Gazebo plugins in ROS", *Gazebosim.org*, 2018. [Online]. Available: http://gazebosim.org/tutorials?tut=ros_gzplugins.

[2.9] "tf/Overview/Transformations - ROS Wiki", *Wiki.ros.org*, 2018. [Online]. Available: http://wiki.ros.org/tf/Overview/Transformations.

[2.10] Available :https://link.springer.com/chapter/10.1007%2F978-3-540-30552-1_4

[2.11] M. Rubenstein and R. Nagpal, "Kilobot: A Robotic Module for Demonstrating Behaviors in a Large Scale (\(2^{10}\) Units) Collective", *Dash.harvard.edu*, 2018. [Online]. Available: https://dash.harvard.edu/handle/1/5504015.

[2.12] "K-Team Corporation – Mobile Robotics", *K-team.com*, 2018. [Online]. Available: https://www.k-team.com/.

[2.13] "Swarm robotics", *Brl.ac.uk*, 2018. [Online]. Available:http://www.brl.ac.uk/research/researchthemes/swarmrobotics.aspx.

[2.14] U. Sheffield, "Advances in swarm robotics (and multi-robot systems) - Available Projects - Postgraduates PhD - ACSE - The University of Sheffield", *Sheffield.ac.uk*, 2018. [Online]. Available: https://www.sheffield.ac.uk/acse/research-degrees/available/rg_3.

[2.15] "Pi Swarm Robot - York Robotics Laboratory, The University of York", *York.ac.uk*, 2018. [Online]. Available: https://www.york.ac.uk/robot-lab/piswarm/.

[3.1] C. Pinciroli, A. Lee-Brown and G. Beltrame, "Buzz: An Extensible Programming Language for Self-Organizing Heterogeneous Robot Swarms", *Arxiv.org*, 2018. [Online]. Available: https://arxiv.org/abs/1507.05946.

[3.2] Available: https://link.springer.com/article/10.1007/s11721-012-0072-5

[3.3] "scrimmage – Multi-Agent Robotics Simulator", *Scrimmagesim.org*, 2018. [Online]. Available: https://www.scrimmagesim.org/.

[3.4] "ROS.org | Powering the world's robots", *Ros.org*, 2018. [Online]. Available: http://www.ros.org/.

[3.5] "Gazebo", *Gazebosim.org*, 2018. [Online]. Available: http://gazebosim.org/.

[3.6] "rviz - ROS Wiki", *Wiki.ros.org*, 2018. [Online]. Available: http://wiki.ros.org/rviz.

[3.7] hokuyo_node - ROS Wiki", *Wiki.ros.org*, 2018. [Online]. Available: http://wiki.ros.org/hokuyo_node.

[3.8] "Create a ROS Sensor Plugin for Gazebo | The Construct", *The Construct*, 2018. [Online]. Available: http://www.theconstructsim.com/create-a-ros-sensor-plugin-for-gazebo/.

[3.9] "OpenSLAM.org", *Openslam.org*, 2018. [Online]. Available: http://openslam.org/gmapping.html.

[3.10] "Husky UGV - Outdoor Field Research Robot by Clearpath", *Clearpath Robotics*, 2018. [Online]. Available: https://www.clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/.

[3.11] "global_planner - ROS Wiki", *Wiki.ros.org*, 2018. [Online]. Available: http://wiki.ros.org/global_planner.

[3.12] "base_local_planner - ROS Wiki", *Wiki.ros.org*, 2018. [Online]. Available: http://wiki.ros.org/base_local_planner.

[4.1] "How to Select your Wi-Fi Module", 2018. [Online]. Available: https://www.testequity.com/documents/pdf/applications/wi-fi-module-an.pdf.

[4.2] "Minimalistic approach towards communication and perception in microrobotic swarms - IEEE Conference Publication", *Ieeexplore.ieee.org*, 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1545594/.

[4.3] Available:https://www.electroschematics.com/wp-content/uploads/2013/07/HCSR04-datasheet-version-1.pdf

[4.4] "Timers on the ATmega168/328 - QEEWiki", *Sites.google.com*, 2018. [Online]. Available:https://sites.google.com/site/qeewiki/books/avr-guide/timers-on-the-atmega328.

[4.5] “VMA409", 2018. [Online]. Available:https://www.velleman.eu/downloads/29/vma409_a4v01.pdf.

[4.6] Available: https://www.pololu.com/file/0J677/optical-encoder-for-micro-metal-gearmotor-schematic-diagram.pdf.

[4.7] "Interfacing Rotary Encoder with AVR Microcontroller (ATmega8)", *Circuitdigest.com*, 2018. [Online]. Available: https://circuitdigest.com/microcontroller-projects/avr-rotary-encoder-interfacing.

[4.8] "ESP8266 12E Range test between 2 esp devices . - Everything ESP8266", *Esp8266.com*, 2018. [Online]. Available:https://www.esp8266.com/viewtopic.php?f=6&t=12456.

[4.9] *Kloppenborg.net*, 2018. [Online]. Available:http://www.kloppenborg.net/images/blog/esp8266/esp8266-esp12e-specs.pdf.

[4.10] "Home - IoTMonk", *IoTMonk*, 2018. [Online]. Available: http://iotmonk.com/.

[4.11] "ATmega88/ATmega168", 2018. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-9365-Automotive-Microcontrollers-ATmega88-ATmega168_Datasheet.pdf.

[4.12] *Ineltro.ch*, 2018. [Online]. Available:https://www.ineltro.ch/media/downloads/SAAItem/45/45958/36e3e7f3-2049-4adb-a2a7-79c654d92915.pdf.

Appendices

[1] Mybot.xacro. Code that describes the simulated robot.

```
<?xml version="1.0"?>

<robot name="robot_model" xmlns:xacro="http://www.ros.org/wiki/xacro">
 <xacro:arg name="nsp" default="h0" />
<xacro:arg name="nsp" default="h0_tf" />


<xacro:property name="PI" value="3.1415926535897931"/>

<xacro:property name="chassisHeight" value="0.1"/>
<xacro:property name="chassisLength" value="0.4"/>
<xacro:property name="chassisWidth" value="0.2"/>
<xacro:property name="chassisMass" value="50"/>

<xacro:property name="casterRadius" value="0.05"/>
<xacro:property name="casterMass" value="5"/>

<xacro:property name="wheelWidth" value="0.05"/>
<xacro:property name="wheelRadius" value="0.1"/>
<xacro:property name="wheelPos" value="0.2"/>
<xacro:property name="wheelMass" value="5"/>

<xacro:property name="cameraSize" value="0.05"/>
<xacro:property name="cameraMass" value="0.1"/>

 <xacro:include filename="$(find mybot_description)/urdf/mybot.gazebo" />
 <xacro:include filename="$(find mybot_description)/urdf/materials.xacro" />
 <xacro:include filename="$(find mybot_description)/urdf/macros.xacro" />


<link name='$(arg nsp)/footprint'/>

<joint name="$(arg nsp)/base_joint" type="fixed">
  <parent link="$(arg nsp)/footprint"/>
  <child link="$(arg nsp)/chassis"/>
</joint>


<link name='$(arg nsp)/chassis'>
  <collision>
    <origin xyz="0 0 ${wheelRadius}" rpy="0 0 0"/>
    <geometry>
      <cylinder length="0.1" radius="0.2"/>
    </geometry>
  </collision>
  <visual>
    <origin xyz="0 0 ${wheelRadius}" rpy="0 0 0"/>
    <geometry>
       <cylinder length="0.1" radius="0.2"/>
    </geometry>
    <material name="orange"/>
  </visual>
  <inertial>
    <origin xyz="0 0 ${wheelRadius}" rpy="0 0 0"/>
    <mass value="${chassisMass}"/>
    <cylinder_inertia m="${chassisMass}" r="0.2" h="0.1"/>
  </inertial>
</link>


<link name="$(arg nsp)/caster_wheel">
  <collision>
    <origin xyz="${casterRadius-chassisLength/2} 0 ${casterRadius-
chassisHeight+wheelRadius}" rpy="0 0 0"/>
    <geometry>
      <sphere radius="${casterRadius}"/>
    </geometry>
  </collision>
```

```
  <visual>
    <origin xyz="${casterRadius-chassisLength/2} 0 ${casterRadius-
chassisHeight+wheelRadius}" rpy="0 0 0"/>
    <geometry>
      <sphere radius="${casterRadius}"/>
    </geometry>
    <material name="red"/>
  </visual>

  <inertial>
    <origin xyz="${casterRadius-chassisLength/2} 0 ${casterRadius-
chassisHeight+wheelRadius}" rpy="0 0 0"/>
    <mass value="${casterMass}"/>
    <sphere_inertia m="${casterMass}" r="${casterRadius}"/>
  </inertial>
</link>

<joint name="$(arg nsp)/fixed" type="fixed">
  <parent link="$(arg nsp)/chassis"/>
  <child link="$(arg nsp)/caster_wheel"/>
</joint>


<wheel lr="right" tY="-1.65"/>
<wheel lr="left" tY="1.65"/>


<link name="$(arg nsp)/sensor_base">
  <collision>
    <origin xyz="0 0 0.2 " rpy="0 0 0"/>
    <geometry>
      <box size="0.1 0.1 0.1"/>
    </geometry>
  </collision>

  <visual>
    <origin xyz="0 0 0.2" rpy="0 0 0"/>
    <geometry>
      <box size="0.1 0.1 0.1"/>
    </geometry>
    <material name="red"/>
  </visual>

  <inertial>
    <origin xyz="0 0 0.2" rpy="0 0 0"/>
    <mass value="1"/>
    <inertia ixx="1" ixy="0" ixz="0" iyy="1" iyz="0" izz="1" />
  </inertial>
</link>

<joint name="$(arg nsp)/sensor_base" type="fixed">
  <parent link="$(arg nsp)/chassis"/>
  <child link="$(arg nsp)/sensor_base"/>
</joint>


  <joint name="$(arg nsp)/light_sensor_joint" type="fixed">
    <axis xyz="0 1 0" />
    <origin xyz="0.15 -0.1 0.2" rpy="0 0 3.14"/>
    <parent link="$(arg nsp)/chassis"/>
    <child link="$(arg nsp)/light_sensor"/>
  </joint>

  <joint name="$(arg nsp)/light_sensor_joint_second" type="fixed">
    <axis xyz="0 1 0" />
    <origin xyz="0.15 0.1 0.2" rpy="0 0 3.14"/>
    <parent link="$(arg nsp)/chassis"/>
    <child link="$(arg nsp)/light_sensor_second"/>
  </joint>



  <!-- Light Sensor A -->
  <link name="$(arg nsp)/light_sensor">
    <collision>
```

```
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
  <box size="0.1 0.1 0.1"/>
    </geometry>
  </collision>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
     <box size="0.1 0.1 0.1"/>
    </geometry>
    <material name="black"/>
  </visual>

  <inertial>
    <mass value="1e-5" />
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
  </inertial>
</link>



<!-- Light Sensor B -->
<link name="$(arg nsp)/light_sensor_second">
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
  <box size="0.1 0.1 0.1"/>
    </geometry>
  </collision>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
     <box size="0.1 0.1 0.1"/>
    </geometry>
    <material name="black"/>
  </visual>

  <inertial>
    <mass value="1e-5" />
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
  </inertial>
</link>


<joint name="$(arg nsp)/hokuyo_joint_0" type="fixed">
  <axis xyz="0 1 0" />
  <origin xyz="0 0 0.28" rpy="0 0 0"/>
  <parent link="$(arg nsp)/sensor_base"/>
  <child link="$(arg nsp)/hokuyo_link_0"/>
</joint>



<!-- Hokuyo Laser -->
<link name="$(arg nsp)/hokuyo_link_0">
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
  <box size="0.1 0.1 0.1"/>
    </geometry>
  </collision>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://mybot_description/meshes/hokuyo.dae"/>
    </geometry>
  </visual>

  <inertial>
    <mass value="1e-5" />
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
```

```
    </inertial>
  </link>

</robot>
```

[2] Mybots_world.launch. Code that launches the hole system.

```
<?xml version="1.0"?>
<launch>
          <include file="$(find gazebo_ros)/launch/empty_world.launch">
                    <arg name="world_name" value="$(find
mybot_gazebo)/worlds/mybot.world"/>
                    <arg name="gui" value="true"/>
          </include>


          <group ns="mybot1">
          <include file="$(find mybot_gazebo)/launch/mybot_world.launch">
          <arg name="namespace" value="mybot1"/>
          </include>
          </group>


<node name="rviz" pkg="rviz" type="rviz"  output="screen"/>

</launch>
```

[3] Mybot_world.launch. Code that launches each individual S.R

```
<?xml version="1.0"?>
<launch>

  <arg name="namespace" default="mybot" />
  <arg name="tfpre" default="$(arg namespace)" />

 <param name="robot_description" command="$(find xacro)/xacro.py '$(find
mybot_description)/urdf/mybot.xacro'
 nsp:=$(arg namespace)" />



 <node name="mybot_spawn" pkg="gazebo_ros" type="spawn_model"
output="screen"
respawn="false"
 args="-urdf
       -param robot_description
       -model robot_$(arg namespace)" />


   <node pkg="mybot_gazebo" type="map_broadcaster"
          name="map_broadcaster">
   <param name="param" value="/$(arg tfpre)/odom"/>
   </node>

   <node pkg="mybot_gazebo" type="changer"
          name="changer"/>



  <!-- convert joint states to TF transforms for rviz, etc -->
  <node name="state_publisher"
        pkg="robot_state_publisher"
        type="robot_state_publisher"
```

```
        respawn="false" output="screen">
    <remap from="/joint_states" to="/$(arg tfpre)/joint_states" />
  </node>

 <rosparam file="$(find mybot_control)/config/mybot_control.yaml" command="load"/>

  <!-- load the controllers -->
  <node name="controller_spawner"
    pkg="controller_manager"
    type="spawner" respawn="false"
    output="screen"
    args="joint_state_controller
      rightWheel_effort_controller
      leftWheel_effort_controller"/>


</launch>
```

[4] Amcl_demo.launch. Launcher for all the navigation stack.

```
<?xml version="1.0"?>
<launch>

    <arg name="scan_topic" value="/mybot1/laser" />
    <arg name="odom_topic" value="/mybot1/odom_diffdrive" />
    <arg name="/use_sim_time" value="true"/>
  <!-- Map server -->
  <arg name="map_file" default="$(find code)/src/maps/mymap.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

   <!--- Run AMCL -->
  <include file="$(find mybot_2dnav)/launch/amcl.launch" />

  <!--- Run Move Base -->
  <include file="$(find mybot_2dnav)/launch/move_base.launch" />


</launch>
```

[5] Amcl.launch. Launcher for the AMCL system

```
<?xml version="1.0"?>
<launch>

  <arg name="use_map_topic"   default="true"/><!--default: false-->
  <arg name="scan_topic"      default="/mybot1/laser"/>
  <arg name="odom_topic"      default="/mybot1/odom_diffdrive"/>
  <arg name="odom_frame_id"   default="mybot1/odom"/>
  <arg name="base_frame_id"   default="mybot1/footprint"/>
  <arg name="global_frame_id" default="map"/>
  <node pkg="amcl" type="amcl" name="amcl">

    <!--<param name="use_map_topic" value="$(arg use_map_topic)"/>-->
    <!-- Publish scans from best pose at a max of 10 Hz -->
    <!--<param name="odom_model_type"           value="diff"/>
    <param name="odom_alpha5"               value="0.1"/>
    <param name="gui_publish_rate"          value="10.0"/>
    <param name="laser_max_beams"             value="60"/>
    <param name="laser_max_range"           value="12.0"/>
    <param name="min_particles"             value="500"/>
    <param name="max_particles"             value="2000"/>
    <param name="kld_err"                   value="0.05"/>
    <param name="kld_z"                     value="0.99"/>
    <param name="odom_alpha1"               value="0.2"/>
    <param name="odom_alpha2"               value="0.2"/>-->
    <!-- translation std dev, m -->
    <!--<param name="odom_alpha3"                value="0.2"/>
    <param name="odom_alpha4"               value="0.2"/>
```

```
    <param name="laser_z_hit"               value="0.5"/>
    <param name="laser_z_short"             value="0.05"/>
    <param name="laser_z_max"               value="0.05"/>
    <param name="laser_z_rand"              value="0.5"/>
    <param name="laser_sigma_hit"           value="0.2"/>
    <param name="laser_lambda_short"        value="0.1"/>
    <param name="laser_model_type"          value="ray"/>-->
    <!-- <param name="laser_model_type" value="beam"/> -->
    <!--<param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d"              value="0.25"/>
    <param name="update_min_a"              value="0.2"/>
    <param name="odom_frame_id"             value="$(arg odom_frame_id)"/>
    <param name="base_frame_id"             value="$(arg base_frame_id)"/>
    <param name="global_frame_id"           value="$(arg global_frame_id)"/>
    <param name="resample_interval"         value="1"/>-->
    <!-- Increase tolerance because the computer can get quite busy -->
    <!--<param name="transform_tolerance"       value="1.0"/>
    <param name="recovery_alpha_slow"       value="0.0"/>
    <param name="recovery_alpha_fast"       value="0.0"/>
    <remap from="scan"                      to="$(arg scan_topic)"/>
    <remap from="odom"                      to="$(arg odom_topic)"/>-->


    <param name="use_map_topic" value="$(arg use_map_topic)"/>
    <!-- Publish scans from best pose at a max of 10 Hz -->
    <param name="odom_model_type" value="diff"/>
    <param name="odom_alpha5" value="0.1"/>
    <param name="gui_publish_rate" value="10.0"/>
    <param name="laser_max_beams" value="60"/>
    <param name="laser_max_range" value="12.0"/>
    <param name="min_particles" value="500"/>
    <param name="max_particles" value="2000"/>
    <param name="kld_err" value="0.05"/>
    <param name="kld_z" value="0.99"/>
    <param name="odom_alpha1" value="0.2"/>
    <param name="odom_alpha2" value="0.2"/>
    <!-- translation std dev, m -->
    <param name="odom_alpha3" value="0.2"/>
    <param name="odom_alpha4" value="0.2"/>
    <param name="laser_z_hit" value="0.5"/>
    <param name="laser_z_short" value="0.05"/>
    <param name="laser_z_max" value="0.05"/>
    <param name="laser_z_rand" value="0.5"/>
    <param name="laser_sigma_hit" value="0.2"/>
    <param name="laser_lambda_short" value="0.1"/>
    <param name="laser_model_type" value="ray"/>
    <!-- <param name="laser_model_type" value="beam"/> -->
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d" value="0.25"/>
    <param name="update_min_a" value="0.2"/>
    <param name="odom_frame_id" value="odom"/>
    <param name="resample_interval" value="1"/>
    <!-- Increase tolerance because the computer can get quite busy -->
    <param name="transform_tolerance" value="1.0"/>
    <param name="recovery_alpha_slow" value="0.0"/>
    <param name="recovery_alpha_fast" value="0.0"/>
    <remap from="scan" to="$(arg scan_topic)"/>
    <remap from="odom" to="$(arg odom_topic)"/>

  </node>

</launch>
```

[6] Main.cpp. Main code for the first robot of the swarm group. Code working with singe individuals but not finished yet for the swarms. Note: comments are not added adequately neither variables set in the correct way. This code still needs to be revised and commented properly. This is just an overview of what it was and what it would be.

```
#include <ros/ros.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <sensor_msgs/LaserScan.h>
#include <geometry_msgs/Twist.h>
#include <sensor_msgs/Range.h>
#include <sensor_msgs/Illuminance.h>
#include "std_msgs/String.h"
#include <cmath>
#include <tf/transform_listener.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <actionlib/client/simple_action_client.h>



float dist = 0, dist_right = 0, dist_left = 0;
float linear = 0.5;
float angular;
int front_beam = 0, right_beam = 0, left_beam = 0;
int data = 0;
int size = 0;
int randoms = 0;
int flag = 0;
float total_right=0, total_left=0;
float lux0 = 0, lux1 = 0;
float light_x[20],light_y[20];
float abslight_x[20],abslight_y[20];
float total[20];
int light_spot_counter[20];
float lightx, lighty;
int take_first = 0,obstacle = 0, flag_goal = 0;
float origin_x = 8.2, origin_y = -0.5, origin_z;
float posex, posey;
float ori[2];
int vel_controller = 0;
int check_new = 0;
int repetido=0;
std::string lightpointx;
std::string lightpointy;
int gotopoint=0;
int sendarrays=0;


typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> MoveBaseClient;



////////////////////////////////    CALLBACKS
/////////////////////////////////////////////////////////////////////////////
void sendinformation(){
      if(sendarrays==1){
       std::stringstream kak;

kak<<light_x[0]<<light_y[0]<<light_x[1]<<light_y[1]<<light_x[2]<<light_y[2]<<light_x[3
]<<light_y[3]<<light_x[4]<<light_y[4]<<light_x[5]<<light_y[5]<<light_x[6]<<light_y[6]<
<light_x[7]<<light_y[7]<<light_x[8]<<light_y[8]<<light_x[9]<<light_y[9]<<light_x[10]<<
light_y[10]<<light_x[11]<<light_y[11]<<light_x[12]<<light_y[12];

        std::string kek="array";
        kek+=kak.str();

        printf("string %s\n",kek.c_str());

       str_msg.data = kek;
     }
     if(sendarrays==0){
     std::stringstream jak;
     jak<<posex<<posey;
     std::string jek="nom";
     jek+=jak.str();
     str_msg.data = jek;
   }
```

```
}
////////////////////////////////     CHATTER
/////////////////////////////////////////////////////////////////

void callback(const std_msgs::String::ConstPtr& msg)
{
  ros::Rate loop_rate(100);
  std::string stored = msg->data;
  std::string origin ("origin");

  std::size_t found = stored.find(origin);
  if (found!=std::string::npos){
  std::string lit = stored.substr(6,12);//(stored.find("<")+1);
  std::string lot = stored.substr(13,19);//(stored.find(">")+1);
  origin_x = std::atof(lit.c_str());
  origin_y = std::atof(lot.c_str());
     }
}
////////////////////////////////   LIGHT SENSORS
/////////////////////////////////////////////////////////////////////////

void callback_l0(const sensor_msgs::Illuminance::ConstPtr& msg)
{
  ros::Rate loop_rate(100);
  lux0 = msg->illuminance;
}

void callback_l1(const sensor_msgs::Illuminance::ConstPtr& msg)
{
  ros::Rate loop_rate(100);
  lux1 = msg->illuminance;
}

////////////////////////////   LASER
/////////////////////////////////////////////////////////////////////////////////////

void callback(const sensor_msgs::LaserScan::ConstPtr& scan)
{
  ros::Rate loop_rate(100);
  total_right= 0;
  total_left = 0;
  size = (scan->angle_max - scan->angle_min)/scan->angle_increment;
  float ranges[size-1];

  for(data=1;data<size;data++){
          ranges[data] = scan->ranges[data];}

  front_beam = (size-1)/2;
  right_beam = front_beam + 240;
  left_beam = front_beam - 240;

  for(right_beam;right_beam!=front_beam;right_beam--){total_right =
total_right+ranges[right_beam];}
  for(left_beam;left_beam!=front_beam;left_beam++){total_left =
total_left+ranges[left_beam];}

  total_right = total_right/240;
  total_left = total_left/240;


 float smallest=1000;  /*order array by size*/
  for ( int i=1;  i < size;  ++i ){
    if ( ranges[i] < smallest ){
      smallest = ranges[i] ;}
  }


  dist = ranges[front_beam];  /*particular distances*/
  dist_right= ranges[right_beam];
  dist_left= ranges[left_beam];



  if( smallest<2.5) //this refers to the laser scan that is just in front of the
robot, the number 360
  {
```

```
   obstacle = 0;
   if((total_left*10)<((total_right*10))){
    angular = -1/total_left *4;}

   else
            {angular = 1/total_right *4;}

  }

  else { obstacle=1;}//velocity is inversely proportional to the distance to the wall
in front

}


////////////////// MOVE TO POSE FUNCTION
/////////////////////////////////////////////////////

void goToPose(float x, float y, float z)
{
  vel_controller = 1;
  MoveBaseClient ac("mybot1/move_base", true);   //tell the action client that we want
to spin a thread by default
  printf("STARTING MOVEMENT\n");

  while(!ac.waitForServer(ros::Duration(5.0))){     //wait for the action server to
come up
    ROS_INFO("Waiting for the move_base action server to come up");}

  move_base_msgs::MoveBaseGoal goal;


  goal.target_pose.header.frame_id = "map";    //we'll send a goal to the robot to
move 1 meter forward
  goal.target_pose.header.stamp = ros::Time::now();

  goal.target_pose.pose.position.x = x ;
  goal.target_pose.pose.position.y = y ;
  goal.target_pose.pose.orientation.z = 1.57 ;
  goal.target_pose.pose.orientation.w = 1 ;
  //goal.target_pose.pose.orientation = odom_quat ;

  ROS_INFO("Sending goal to %f, %f, %f
\n",goal.target_pose.pose.position.x,goal.target_pose.pose.position.y,z);
  ac.sendGoal(goal);


  ac.waitForResult();

  if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED){
    ROS_INFO("Task completed"); vel_controller = 0; }
  else{
    ROS_INFO("Task failed"); vel_controller = 0; }
}

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////


int main(int argc, char **argv)
{



  std::ifstream collected;
  ros::init(argc, argv, "main");

  ros::NodeHandle n;

  ros::Publisher pub = n.advertise<geometry_msgs::Twist>("/mybot1/cmd__vel", 100);
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
  ros::Subscriber sub = no.subscribe<std_msgs::String>("chatter",1000,callback);

  ros::Subscriber sub =
n.subscribe<sensor_msgs::LaserScan>("/mybot1/laser",1000,callback);
  ros::Subscriber sub_light0 =
n.subscribe<sensor_msgs::Illuminance>("mybot1/lightSensor",1000,callback_l0);
```

```
  ros::Subscriber sub_light1=
n.subscribe<sensor_msgs::Illuminance>("mybot1/lightSensor_second",1000,callback_l1);
  ros::Duration(3).sleep();
  tf::TransformListener listener;

  ros::Rate loop_rate(100);

  geometry_msgs::Twist cmd_msg;
  std_msgs::String str_msg;
  int count=0;
  light_x[0]=1;
  std::fstream file;

  while (ros::ok()){


    tf::StampedTransform transform;

    float luxavg = (lux0+lux1)/2;
//////////////////////////////////// SAVE POSITION WHEN LIGHT FOUND
////////////////////////////////////////////////////////////////

    if(take_first==0 || lux0>150 || lux1>150){
      try{
        ros::Time now = ros::Time(0);
        listener.waitForTransform("map","mybot1/chassis",now,ros::Duration(3.0));
        listener.lookupTransform("map", "mybot1/chassis",now, transform);}


     catch (tf::TransformException ex){
        ROS_ERROR("%s",ex.what());
        ros::Duration(1.0).sleep();}

     if(take_first==0){
       take_first=1;
       origin_x = transform.getOrigin().getX();
       origin_y = transform.getOrigin().getY();
       origin_z = transform.getOrigin().getZ();

       ori[0] = origin_x;
       ori[1] = origin_y;

       std::stringstream lol;
       lol<<origin_x<<origin_y;




        std::string lel="origin";
        lel+=lol.str();

        printf("string %s\n",lel.c_str());

       str_msg.data = lel;


      printf(" oringin, x: %f, y:%f, z:%f \n",origin_x, origin_y,origin_z);}

     else{
                    lightx = transform.getOrigin().getX();
       lighty = transform.getOrigin().getY();
              printf("something found at lightx: %f, lighty: %f\n", lightx,lighty);
       for(int i=0;i<20;i++){
                if(light_x[i]==1){
                   for(int k=0; k<20; k++){
                     if((lightx<(light_x[k]+0.5) && lightx>(light_x[k]-0.5)) ||
(lighty<(light_y[k]+0.5) && lighty>(light_y[k]-
0.5))){repetido=1;light_spot_counter[k]++;}
            }

           if (repetido==0){
                    printf("NEW LIGHT SPOT\n");
                     light_x[i] = lightx;
                     light_y[i] = lighty;
                     printf("Light number %d , at x: %f, y:
%f\n",i,light_x[i],light_y[i]);
```

```
            for(int z=0;z<20;z++){
              std::stringstream ss;
              std::stringstream ll;
              ss<<light_x[i];
              ll<<light_y[i];
              lightpointx+=ss.str();
              lightpointx+=",";
              lightpointy+=ll.str();
              lightpointy+=",";
             }
            lightpointx+="|||";
            lightpointx+=lightpointy;
            lightpointx+="___";
            file.open("/home/mateo/output.txt",std::fstream::app);
            file << lightpointx;
            file.close();

            light_spot_counter[i]++;
            goToPose(origin_x,origin_y,origin_z);
            sendinformation();
            gotopoint=1;
                  }

if(repetido==1){goToPose(origin_x,origin_y,origin_z);sendarrays=1;sendinformation();
sendarrays=0; gotopoint=1;}

                                  }
        }

       repetido=0;
       for(int i=0;i<20;i++){
         if((light_x[i]!=0 && light_x[i]!=1) && (light_x[i+1]==0)){light_x[i+1]=1;}
       }
      }
    }


    else{

      try{
        ros::Time now = ros::Time(0);
        listener.waitForTransform("map","mybot1/chassis",now,ros::Duration(3.0));
        listener.lookupTransform("map", "mybot1/chassis",now, transform);}


     catch (tf::TransformException ex){
        ROS_ERROR("%s",ex.what());
        ros::Duration(1.0).sleep();}

      posex = transform.getOrigin().getX();
      posey = transform.getOrigin().getY();
      posex=std::abs(posex);
      posey=std::abs(posey);
      sendinformation();
      int exists=0;

      if(gotopoint==1){

       for(int o=0;o<20;o++){
         if(light_x[o]!=0 && light_x[o]!=1){
          exists=1;
           abslight_x[o]=std::abs(light_x[o]);
           abslight_y[o]=std::abs(light_y[o]);
           total[o]=(posex-abslight_x[o])+(posey-abslight_y[o]);
          }
        }
        float small=1000;
        int point=0;
        for(int p=0;p<20;p++){
          if(total[p]<small && total[p]>1){small=total[p];point=p;}
        }

       if(exists==1){
         float gotox = light_x[point];
         float gotoy = light_y[point];

          printf("light point %f\n",light_x[point]);
```

```
          printf("point %d \n",point );

          goToPose(gotox,gotoy,0);
          gotopoint=0;
         }
       }


      else{
       if(obstacle==1){
         if(lux0<lux1){angular=0.5;}else{angular=-0.5;}
        }
        cmd_msg.angular.z = angular*2;
        cmd_msg.linear.x = linear*2;
        if(vel_controller==0){
        pub.publish(cmd_msg);}
       }
     }


     chatter_pub.publish(str_msg);

     ros::spinOnce();
     loop_rate.sleep();
   }
   return 0;
}
```

Hardware.cpp. Code that controls real robots on the ATMEGA168 programmed using Atmel studio.

```
/*
 * Sensors.c
 *
 * Created: 15/02/2018 16:00:39
 * Author : Mateo
 */

#include <avr/io.h>
#include <avr/interrupt.h> //for interrupts, sei()
#include <avr/pgmspace.h>
#include <string.h>
#include <stdio.h>

#define F_CPU 1000000 //freq to 1MHz

#include <util/delay.h> //for delays
#include <stdlib.h>
#include <stdint.h>       // needed for uint8_t


#define BUAD    1200
#define BRC     ((F_CPU/16/BUAD) - 1)
#define TX_BUFFER_SIZE  128

char serialBuffer[TX_BUFFER_SIZE];
char enfe[15];
char a[5];
char b[5];
char c[5];
char d[5];
char info[25];
uint8_t serialReadPos = 0;
uint8_t serialWritePos = 0;

void appendSerial(char c);
```

```
void serialWrite(char  c[]);


char cw1,cw2;

float outM1A, outM1B,outM2A,outM2B, outA_new, outB_new;
int counter1=0, counter2=0, c1 = 0, c2 = 0;
float velocity1, velocity2;
int distance1=0, distance2=0;
float momentary_distance1, momentary_distance2;
int count_time1=0, count_time2;

int interruptnumber = 0;

int i=0, j=0;
int flag_turn=0;
int COUNTB_old =0;
int COUNTA_old =0;

static volatile int pulse = 0, pulse0;//variable for the pulse distance


int main(void)
{
        DDRB = 0b11111110; //setting all Port B as output for the motors
        DDRD = 0b11110011; //All but PD2 PD3 PD4 as outputs, echo is the input
        DDRC = 0x00; //Encoders an inputs, maybe only set PC0-5
    PORTC=0b00111111; //Pull-Up in pins 0 to 5
        _delay_ms(50);

    EICRA |= (1 << ISC10) | (1<<ISC00); // set INT0 INT1 to trigger on ANY logic change
    EIMSK |= (1 << INT1) | (1<<INT0);// | (1<<INT1);

 //Timer/Counter Control Register 1 A in normal mode
        TCCR1A = 0;


        UBRR0H = (BRC >> 8);
        UBRR0L =  BRC;

        UCSR0B = (1 << TXEN0)  | (1 << TXCIE0);
        UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);

        int16_t COUNTA = 0; //counter
        int16_t COUNTB = 0; //counter

        sei(); //enable interrupts

        while(1)
        {
                int lol = velocity1*10000;
                int lel = momentary_distance1*10;

            count_time1++;
                count_time2++;

                PORTD|=(1<<PIND6);_delay_us(15);PORTD &=~(1<<PIND6);
                sprintf(enfe,"%d\n ",pulse);
                interruptnumber=1;
                cli();
                sei();
                PORTD|=(1<<PIND5);_delay_us(15);PORTD &=~(1<<PIND5);
                sprintf(info,"%d\n ",pulse0);
                interruptnumber=0;

                outM1A= PINC & 0b00100000;
                outM1B= PINC & 0b00010000;
                outM2A= PINC & 0b00001000;
                outM2B= PINC & 0b00000100;

                COUNTB_old=atoi(info);
                COUNTA_old=atoi(enfe);

                COUNTA = COUNTA_old/58;   //count in cm
```

```
                COUNTB = COUNTB_old/58;
                sprintf(info,"%d, %d, %d, %d . ",COUNTA,COUNTB,distance1,distance2);


                serialWrite(info);

                _delay_ms(200);

                cli();
                sei();

                if((COUNTA< 25)|| (COUNTB<25))
                {
                if((COUNTA<5)||(COUNTB<5))
                {
                        if((COUNTA*10)<=(COUNTB*10)) //ERA B
                        {
                                //PORTB=0b01000010; //move left
                                _delay_ms(100);
                        }
                        else
                        {
                                //PORTB=0b10000100; //move right
                                _delay_ms(100);
                        }
                }
                else{
                if((COUNTA*10)<=(COUNTB*10))
                {
                //PORTB=0b00000010; //move left
                _delay_ms(100);
                }
                else
                {
                //PORTB=0b10000000; //move right
                _delay_ms(100);
                }
                }}

                else
                {
                //PORTB=0b10000010; //move forward
                _delay_ms(100);
                }

// ////////////////            ENCODERS
        ////////////////////////////////////////////////////////////////////////
//
                if(outM1A==outM1B){cw1=1;}
                else{cw1=0;}

                if(outM2A==outM2B){cw2=1;}
                else{cw2=0;}

                if(outM1A>20){counter1++;c1++;} //31 seems to be the limit although it might
vary
                momentary_distance1=c1*0.22; //distance in mm per count

                if(counter1>=150)               //equivalent to 33mm
                {
                        counter1=0;
                        distance1=distance1+33;}

                if(count_time1>=10)                 //this gets the velocity
                {
                velocity1=(momentary_distance1/50);  //mm/ms = m/s
                momentary_distance1=0;              //reset parameters
                count_time1=0;
                c1=0;}



                if(outM2A==outM2B){cw2=1;}
                else{cw2=0;}
```

```
                if(outM2A==outM2B){cw2=1;}
                else{cw2=0;}

                if(outM2A>5){counter2++;}
                momentary_distance2=counter2*0.22; //distance in mm per count

                if(counter2>=150)               //equivalent to 33mm
                {
                        counter2=0;
                distance2=distance2+33;}

                if(count_time2>=10)                  //this gets the velocity
                {
                        velocity2=(momentary_distance2/50);  //mm/ms = m/s
                        momentary_distance2=0;              //reset parameters
                count_time2=0;}


        }
}

ISR(INT0_vect)  // INT0 interrupt function cli(); is dissable interrupts
{
        if(interruptnumber==0){
        if (i==1)
        {

                TCCR1B=0;    //Timer/Counter Control Register 1 B in normal mode
                 //storing the counter value in variable pulse from the register that stores
it (16 bit register WHICH IS A TIMER)

                pulse=TCNT1;
                TCNT1=0;     //WHAT?
                i=0;

        }
        if (i==0)
        {
                TCCR1B|=(1<<CS10); //no pre-scaling and start timer
                i=1;               //change to 1
        }
        }
}

//
ISR(INT1_vect)  // INT0 interrupt function cli(); is dissable interrupts
{
  if(interruptnumber==1){
        if (j==1)
        {
                TCCR1B=0;    //Timer/Counter Control Register 1 B in normal mode
                pulse0=TCNT1; //storing the counter value in variable pulse from the register
that stores it (16 bit register WHICH IS A TIMER)
                TCNT1=0;     //WHAT?
                j=0;         //change to 0
        }
        if (j==0)
        {
                TCCR1B|=(1<<CS10); //no pre-scaling and start timer
                j=1;               //change to 1
        }}
        else{pulse0=pulse0;}
}
//



void appendSerial(char c)
{
    serialBuffer[serialWritePos] = c;
    serialWritePos++;

    if(serialWritePos >= TX_BUFFER_SIZE)
```

```
    {
        serialWritePos = 0;
    }
}

void serialWrite(char c[])
{
    for(uint8_t i = 0; i < strlen(c); i++)
    {
        appendSerial(c[i]);
    }

    if(UCSR0A & (1 << UDRE0))
    {
        UDR0 = 0;
    }
}

ISR(USART_TX_vect)
{
    if(serialReadPos != serialWritePos)
    {
        UDR0 = serialBuffer[serialReadPos];
        serialReadPos++;

        if(serialReadPos >= TX_BUFFER_SIZE)
        {
            serialReadPos++;
        }
    }
}
```

Insert.php. Code for inserting a new row in the sensors database for the sonar and odometry data.

```
<?php

header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF8");

//create array for JSON response
$response =  array();

//check if got field from user
if(isset($_GET['sonarright']) && isset($_GET['sonarleft']) && isset($_GET['odomright']) && isset($_GET['odomleft'])){

        $sonarright = $_GET['sonarright'];
        $sonarleft = $_GET['sonarleft'];
        $odomright = $_GET['odomright'];
        $odomleft = $_GET['odomleft'];

//Include connect class
```

```
        $filepath = realpath(dirname(__FILE__));
        require_once($filepath."/db_connect.php");

//connect to database
        $db = new DB_CONNECT();
//FIRE sql query to inser data in database
        $result = mysql_query("INSERT INTO sensors(sonarright,sonarleft,odomright,odomleft)
VALUES('$sonarright','$sonarleft','$odomright','$odomleft')");
//check success
        if($result){
                $response["success"] = 1;
                $response["message"] = "Sensors felt";

//show json response
                echo json_encode($response);
        }
        else{
                //failed to insert data
                $response["success"] = 0;
                $response["message"] = "oh shit";
                echo json_encode($response);
        }
}
else{
                //if parameters are missing
                $response["success"] = 0;
                $response["message"] = "missing parameters, check request";
                echo json_encode($response);
}
?>
```