CG2111A Engineering Principle and Practice II

Semester 2 2022/2023

“Alex to the Rescue”

Final Report

Team: B01-6A

Name	Student No.	Sub-Team	Role
Arun Gandhi Shyam Krishna	A0266505A	Software	Code Implementation
Azfarul Matin Bin Mohamad Afandi	A0254881W	Firmware	Assembly Documentation
Lu Bingyuan	A0259353X	Hardware	Assembly
Shan YuXuan	A0264661B	Software	Calibration

Section 1 System Functionalities

Function	Explanation
Remotely controlled	Alex can be controlled remotely using a separate device (in our case, laptop), the commands will be sent as packets through the local network to the Raspberry Pi (RPi) using the TCP/IP with a secure connection (encrypted). The master control program (MCP) on the RPi will then translate the operator's commands into movement control signals for the connected Arduino board using serial communication. The Arduino will then send signals to the motor driver. The network communication will be done through ROS.
Movement	Alex will be able to carry out the following movement commands:: ● Move forward (2 possible magnitudes) ● Reverse (2 possible magnitudes) ● Turn left / right (precalibrated values) Using the motor encoders attached, higher resolution is achieved.
Environment mapping	Alex will be able to detect its environment by continuously scanning the room with its LiDAR using SLAM technology (Hector SLAM) and send data back to the RPi which can then map out the room using a GUI on a 2D graph. It should be able to compute information such as the distance travelled and relative position to help prevent collision with walls and allow the operator to navigate the environment.
Colour Sensing	Alex will use LEDs and the light sensor to detect the colour of objects in the room.
Extra functionality	● Visualising Alex in rviz by forming axes at the borders. ● Smart control of Alex's movements using keys.

Section 2 Review of State of the Art

2.1 Taurob Tracker

The Taurob Tracker is a tele-operated search and rescue robot. It is made to withstand explosive situations and it is resistant and waterproof too. The functionality of this robot is obstacle and human detection, path planning, and area mapping. (Gerald, 2014). For the hardware aspect, this robot has a front and rear 3D camera and a CMOS camera, an elevated 3D LiDAR, a robotic arm, GPS, and an Intel I7 2.90 GHz desktop computer. As for the software aspect, it is operating using the Ubuntu system. (Georg and Wilfried, 2020)

The advantage of this robot is that it has good manoeuvrability through terrains. The camera also helps the operator to identify the area. Also, the LiDAR allows the robot to give accurate area mapping too. However, this robot has a short operating time due to the use of cameras and the intel i7 computer, which also means that it has a high cost to build.

2.2 Carnegie Mellon Modular Snake Robot

The modular snake robot developed by Carnegie Mellon University is a highly manoeuvrable robot that aims to be used for search and rescue missions. This robot is able to gain access to places easily, such as climbing up a tree or even swimming underwater. (Kelion, 2013)

For the hardware aspects, the segments are connected by flexible links which allows the robot to move flexibly. A joystick can also be used to control the robot. It is also equipped with cameras with LED spotlights. With appropriate skins, the robot can also be operated underwater. Whereas for software, it uses both a control system and a perception system. The control system is used to generate motion patterns for the robot's movements, while the perception system is used to gather information on the environment of the robot by using external devices like cameras (Popular Mechanics, 2009) The advantage of this robot is that it is very flexible which allows it to move around difficult environments. However, it is a very complex build which means that the robot could cost very high considering the amount of research and project needed. Other than that, it might have limited speed despite its agility and flexibility.

Section 3 System Architecture

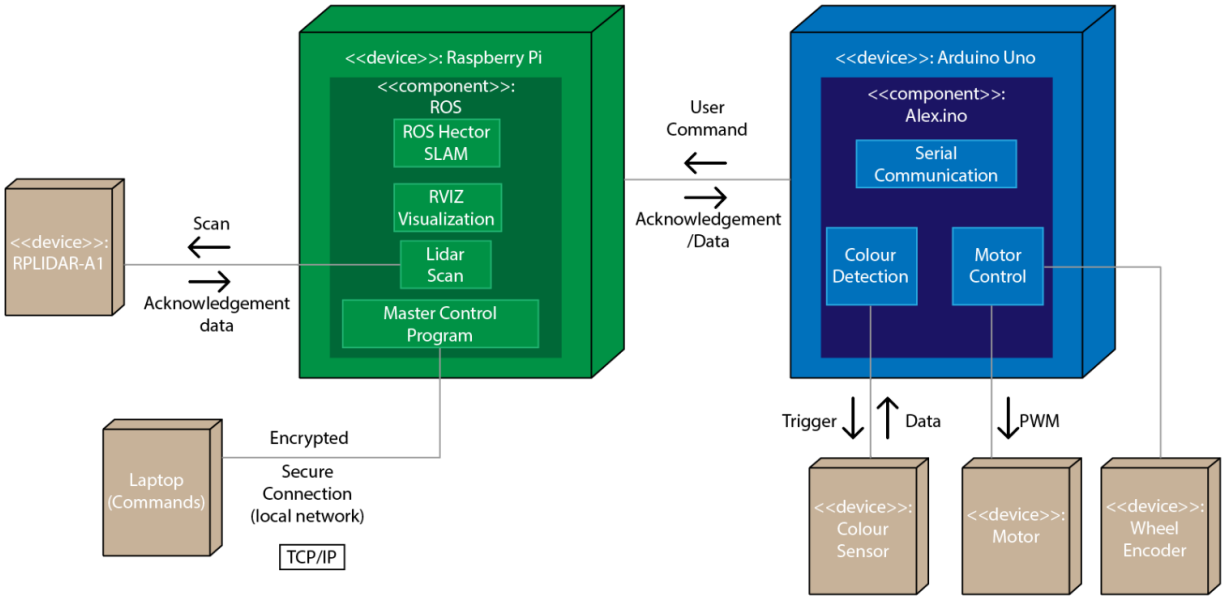


Figure 1: UML Diagram of Alex's system architecture

This section covers the system architecture of Alex. Figure 1 shows the devices and components used in our project.

3.1 Main Architecture

As shown in Figure 1, the two big boxes represent the Arduino UNO and Raspberry Pi. They are the two main devices of Alex's overall architecture. The small brown boxes represent external devices used.

3.2 Arduino UNO

The main purpose of the arduino unit in this project is to

- Control the motors
- Sense colour accurately
- Send the no. of revolutions for both the wheels to the Pi.

Motor control or specifically motor speed control is done by altering the motor's RPM by sending an appropriate PWM signal. Colour sensing is performed by triggering and reading colour data from a colour sensor. By reading data from the wheel encoder and reference against calibrated values of wheel circumference, the Arduino can then calculate the distance travelled by the Alex robot.

The arduino also continuously reads data from the UART port. When a command packet is passed from the UART to the Arduino, it is deserialized and handled

according to the header file. The commands are then executed on the corresponding components of the Arduino, after which an acknowledgement (OK packet) is returned to the RPi if execution is successful. The user can also send commands requesting for data from the Arduino components, such as telemetry and colour reading. This data will be sent via UART back to RPi and via TLS to the user's laptop.

3.2.1 GY-31 Colour Sensor

The GY-31 colour sensor works by shining light onto the target object and detecting the frequency of RGB colours reflected back. The light reflected from the object is directed onto the photodiodes and the intensity of each colour is measured using the RGB filters present in front of the sensor. The frequency measurements are mapped to the RGB values from 0 to 255.

3.2.2 Motor Controller

The Arduino can control motor speed of the motors using PWM. By varying the duty cycle of a PWM signal, the amount of power delivered to the load (the motor) can be varied, causing it to spin at varying RPMs or different directions. In this case, the user instruction is used to generate a suitable PWM duty cycle before being sent to the motors.

3.2.3 Wheel Encoder

The purpose of the wheel encoder is to calculate the distance moved by the robot. The encoder works on the principle of the Hall effect. When the flux change around it is above a threshold value, it produces an interrupt to the Arduino Uno, which enables us to calculate the number of interrupts triggered, ultimately allowing us to measure the number of revolutions using the pre-calibrated values.

3.3 Raspberry Pi

The Raspberry Pi (RPi) uses the TCP/IP protocol which makes it possible to connect external devices wirelessly. It can act as a Transport Layer Security (TLS) server with authentication mechanisms to encrypt communication with the user’s laptop. Therefore, it acts as the medium of communication between the user’s laptop and Arduino Uno via UART protocol.

The RPi acts as the central control for Alex, controlling the motors through sending commands to the Arduino, receiving data and mapping out the room using SLAM and LiDAR and receiving colour sensing data from the Arduino.

The RPi is also where roscore will be launched at and it contains all the files and programs in Alex.

3.3.1 LIDAR

The LiDAR is a TOF (Time of flight) sensor, which scans the environment and measures distances of obstacles present in it. This data is fed to a Hector Simultaneous Localization and Mapping (SLAM) algorithm to create a map of the scanned environment while localising the Alex robot in it. This map and localization is visualised using Rviz which displays data as point clouds, indicating the presence of landmarks and obstacles around the Alex robot.

3.3.2 ROS Nodes involved

ROS (Robot Operating System) Noetic is installed in the Raspberry Pi 4. The *rplidar node* broadcasts the readings from the RPLidar. Then the *slam node* performs the SLAM algorithm, which is followed by the *rviz node* to visualise the data from the RPLidar as a map.

Section 4 Hardware Design

4.1 Alex Overall Design

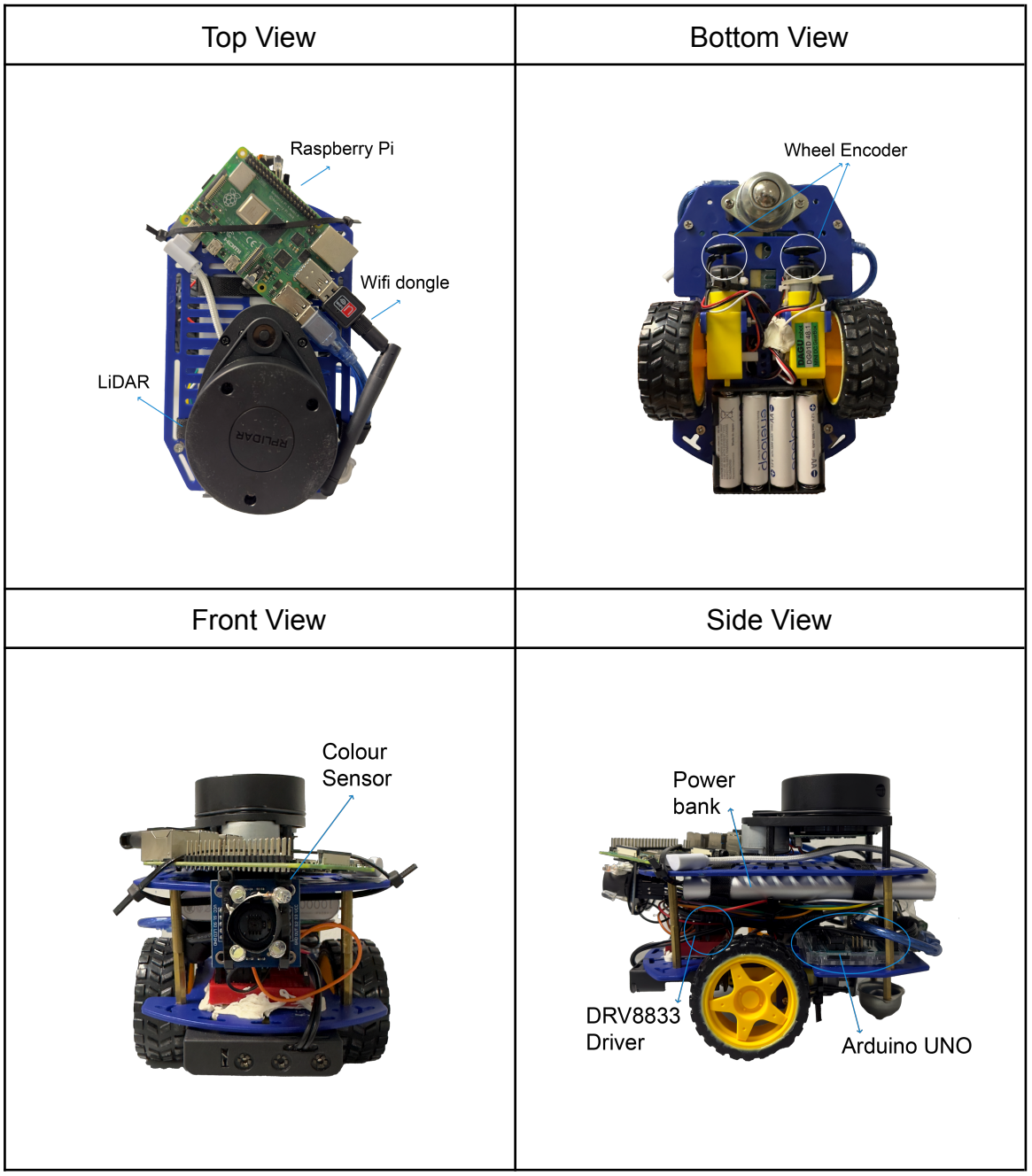


Figure 2: Images of Alex in four different views to show the placement of components

4.2 Placement of Components

The main components of our robot consists of the LiDAR, Raspberry Pi, Arduino UNO, power bank, batteries, wheels, motors and wheel encoder. We have placed the components carefully as we considered factors such as the centre of gravity

and the limitations of the components.There are three layers in our robot, the top, middle, and bottom layer.

4.2.1 Top Layer

The top layer consists of the LiDAR and Raspberry Pi, along with the wifi dongle. The LiDAR is mounted at the top as we did not want any other components to block and interfere when the LiDAR is mapping. The RPi is also placed at the top to ensure that when it overheats, it does not affect any other components nearby as cautioned by our professor. It is also placed at the top so that the wifi dongle will be able to give us a better connection with our laptops.

4.2.2 Middle Layer

The middle layer consists of the GY-31 colour sensor, power bank, Arduino UNO, the DRV8833 dual H-bridge motor driver IC along with the mini breadboard and wires. The colour sensor is strapped in front of the robot right under the top chassis as it is a good height to read a colour when a possible victim is detected. The power bank is strapped under the top chassis to ensure that the centre of gravity of the robot is equal. The breadboard and arduino is placed in the middle layer as it is the safest layer for us to keep our wires protected. It reduces the exposure of the wires, which should be prevented as it counts as a hit and it also helps us to reduce the chances of disconnecting our wires by accident.

4.2.3 Bottom Layer

Lastly, the bottom layer consists of the wheels, motors, wheel encoders and the 4 AA batteries with the holder. The batteries are mounted at the bottom so that we can easily remove and place the batteries back. The wheel encoders are also placed at the bottom as it has to be near the motors to measure the number of revolutions. This is also why the arduino is placed in the middle layer as it prevents the wires from the wheel encoders to the arduino from dangling at the bottom.

Section 5 Firmware Design

5.1 A High-Level Algorithm

5.1.1 Arduino Initialization

- The Arduino is set up with its interrupts and ports initialized.
- The Arduino waits for communication to be established with the RPi at a baud rate of 9600.

5.1.2 Reception of Commands

- The Arduino polls the RPi for the command packet.
- The serialized command packet data is deserialized.
- If the transfer of data was unsuccessful, a bad response packet is returned to indicate the error. Otherwise, an OK packet is returned.
- Commands in the packet are identified by comparing the obtained values with the predeclared constants in the header files.

5.1.3 Execution of Commands

- Once a fair command is recognized, PWM signals are transmitted from the motor control pins according to the values received in the command packet.
- Repeat step 2 until the connection to the RPi is disconnected or Alex is turned off.

5.2 Structure of the Command Packet

char packetType	1 byte	Describes the type of the packet like Command_packet, Response_packet, Hello_packet etc.
char command	1 byte	The instruction to be executed.
char dummy[2]	2 bytes	Padding of 2 bytes to make up 4 bytes
char data[32]	32 bytes	String data
uint32_t params[16]	64 bytes	The values that are required for the execution of the command.

Section 6 Software Design

6.1 High Level Steps:

1. Initialization
2. Transmit and receive commands
3. Executing the commands
4. Repeat step 2 until the navigation is over

6.2 Further Breakdown

6.2.1 Initialization

1. Enable TLS connection
    - User SSH into RPi
    - Initialise TLS server on RPi, connect to Arduino via UART and waits for connection from user
    - Laptop initialises TLS client and search for connection
    - Laptop connects to RPi via TLS
    - RPi and Arduino performs handshake via sending packets
2. Initialize ROS nodes and SLAM
    - Set up ROS Master on Pi
    - Launch the rplidar_node and hectormapping_node on RPi, which triggers the Lidar and publishes the data on the points of intersection.
    - Identify the surroundings and the relative location of Alex using the Hector SLAM algorithm.
    - Send the mapped data of the environment to the user, which is visualised in the rviz environment.

6.2.2 Transmit and receive commands

- Based on the obstacles in Alex's environment, the user would send an appropriate movement command packet and search for the victim.
- Laptop sends a serialised command packet via TLS connection to RPi and it reads the packet.
- RPi checks packet and data for validity.
- If a valid command is detected, RPi serialises it and sends it via UART to the Arduino. If the packet is not valid, RPi sends back a BAD_COMMAND packet.

6.2.3 Executing the commands

- There are two possible commands for forward and backward motion. One commands to move a shorter distance and another for a longer distance. The power utilised would be predetermined.
- For both forward and backward motion, the power utilised by both the motors and the direction of rotation are the same to have a straight motion.

- Similarly, for angular motion, there are 2 commands for each direction which have a different magnitude of rotation.
- For angular motion, the motors are to be rotated in opposite directions to not have any linear displacements.
- The movement speed must be controlled to not exceed the maximum movement speed allowed by the Hector SLAM.

6.3 Additional Software Improvements

6.3 1 Robot Operating System

Robot Operating System (ROS) is a widely-used open-source framework for building and controlling robots. ROS provides a set of tools and libraries that help developers create complex robotic systems more easily by providing a modular and distributed architecture. In ROS, a node is a process that performs a specific computation or task, such as reading sensor data and performing further computations with the data. Nodes can communicate with each other by sending and receiving messages over topics. By using topics, nodes can communicate with each other without needing to know the specific details of how messages are transmitted or received.

6.3.2 Hector Slam

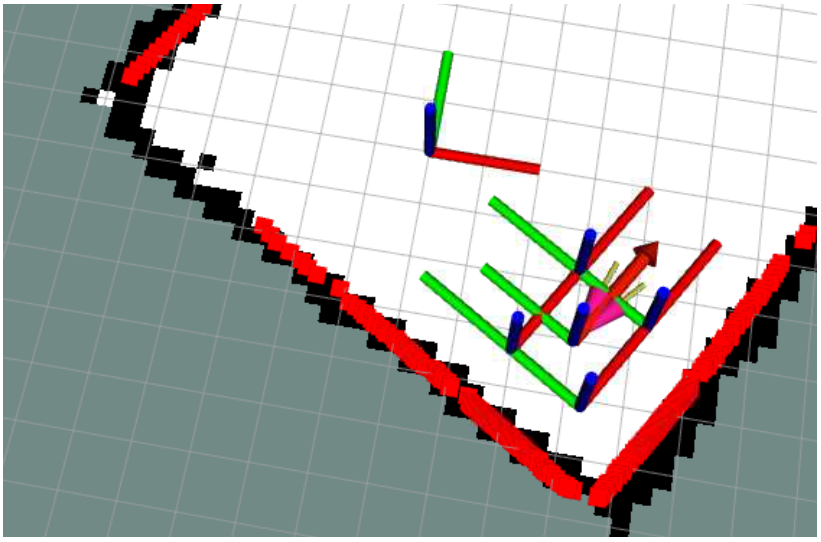
Hector SLAM is a Simultaneous Localization and Mapping (SLAM) algorithm used in Robot Operating System (ROS) to create maps of unknown environments and simultaneously estimate the pose (position and orientation) of a robot within that environment. The Hector SLAM algorithm takes in laser scan data published be /rplidar_node and creates a 2D occupancy grid map of the environment. It uses a scan matching algorithm to align the laser scans with the map and estimate the robot's pose. The algorithm also uses a probabilistic approach to build the map, where it estimates the occupancy probability of each cell in the map based on the laser scan data. Hector SLAM has several advantages over other SLAM algorithms. First, it does not require a GPS or an inertial measurement unit (IMU) to estimate the robot's pose. This makes it simpler to use. Second, it is fast and can create maps in real-time.

6.3.3 Rviz Configuration

The SLAM data is visualised using the tool in ROS called rviz. We customised the packages used and imported our own packages to have better control over Alex. Our customization involved:

- Choosing appropriate cell size, transparency , number of cells
- Resolution of obstacle mapping

- By using the transform (TF) package, we created duplicate axes at a relative position to the base_link to depict the boundary of Alex. We also extended this package to show arrows to depict the protruding colour sensor. These features help us to avoid obstacles and stop Alex at an appropriate distance infront of the victims.



6.3.4 Command Line Interface

Giving the values of magnitude and power for each command is troublesome, less efficient and error prone. So we decided to use 2 sets of keys to control the motion of Alex, which are WSDA and IKLJ. The former is of lower magnitude compared to the latter. The magnitude and powers of each command is calibrated separately considering the smoothness of the HectorSlam and the hump in the maze. One notable feature is that, since we do not give in the magnitudes when performing commands, the magnitudes of turning left and right are set differently. Thus, we could achieve high resolution in controlling the angle of alex. The whole aim of these features is to make the control more efficient, so we used the getche() function in the conio.h library to remove the usage of enter key after pressing the command key everytime.

Section 7 Lessons Learnt - Conclusion

7.1 Centre of Gravity of Alex

One of the greatest mistakes we have made was not taking into consideration the centre of gravity of the robot. Initially, we placed the RPLiDAR in front of the Raspberry Pi at the same top layer. However, we realised that as Alex was attempting to go over the hump, it started to lose its balance and this caused our robot to hit any nearby obstacle or wall that was near the hump. This issue was caused by the unequal weight distribution of the robot. As the LiDAR was significantly heavier than the RPi, Alex was too front heavy. Also, since our caster's wheels are placed at the back of Alex, it was not easy for Alex to travel in a straight position.

Hence, how we solved this issue was to swap the placements of the LiDAR and the RPi, as shown in the top view in figure 2. By doing this, we ensure that the weight of Alex was not too front heavy which caused Alex to topple over when attempting to travel over the hump. Other than the swapping of these two components, we took the time to carefully think of where other components should be placed too such as the placement of the AA batteries and the power bank.

7.2 Non-standardized internal friction of motors

When we setup our code for the movements of Alex, we came across a problem. The linear motion was not straight enough. We thought that there might be an issue with the encoders, so we replaced the old ones. But the problem persisted. After repeated testing we concluded that the inaccuracies were due to unequal and variable internal friction of the left motor. We realised that it was too late to replace the faulty motor.

Thus we tried to fix the problem using the code. Different PWM values were given to the motors for the same input to counter the inaccuracies. We calibrated the new PWM value relative to the normal value for each command. We were able to solve the issue with this solution. We learnt that it is important to ensure that every component of the robot is in proper condition before starting to use them since in some cases, where many components could be involved unlike this, it would be hard to find the component which is causing the trouble.

Appendix

Code snippet for transforming axes:

```
<node pkg="tf" type="static_transform_publisher" name="link1_broadcaster" args="0 0 0 0 0 0 base_link laser 100" />
<node pkg="tf" type="static_transform_publisher" name="alex_tl" args="0.105 0.07 0 0 0 0 base_link TL 100" />
<node pkg="tf" type="static_transform_publisher" name="alex_tr" args="0.105 -0.08 0 0 0 0 base_link TR 100" />
<node pkg="tf" type="static_transform_publisher" name="alex_bl" args="-0.08 0.07 0 0 0 0 base_link BL 100" />
<node pkg="tf" type="static_transform_publisher" name="alex_br" args="-0.08 -0.08 0 0 0 0 base_link BR 100" />
<node pkg="tf" type="static_transform_publisher" name="alex_cll" args="0.16 0.03 0 0 0 0 base_link CL 100" />
<node pkg="tf" type="static_transform_publisher" name="alex_clr" args="0.16 -0.03 0 0 0 0 base_link CR 100" />
```

Code snippet to counter faulty motor:

```
void reverse(float dist, float speed, int a)
{
  dir = BACKWARD;
  int val1, val2;
  if (a = 0){
    val1 = pwmVal(speed);
    val2 = pwmVal(speed*1.3);
  }else{
    val1 = pwmVal(speed);
    val2 = pwmVal(speed*1.03);
  }
```

References

Steinbauer-Wagner, Gerald. (2014). TEDUSAR White Book - State of the Art in Search and Rescue Robots. 10.13140/RG.2.1.4086.9361. https://www.researchgate.net/figure/Taurob-Tracker-Photo-credit-Taurob-GmbH-Taurob-Tracker_fig2_304987927

Georg A. Novotny and Wilfried Kubinger (2020) Design and Implementation of a Mobile Search and Rescue Robot https://openlib.tugraz.at/download.php?id=5f6b0645e51ff&location=browse

Popular Mechanics (2009) Carnegie Mellon Modular Snake Robots for Cave Rescue . . . and Mars? Hands-on First Look https://www.popularmechanics.com/technology/robots/a3684/4285289/

Kelion, B. L. (2013b, April 29). *Snake robot teams up with search-and-rescue dog in US*. BBC News. https://www.bbc.com/news/technology-22340218