AUTONOMOUS ELECTRIC WHEELCHAIR FOR CHILDREN WITH PHYSICAL CHALLENGES

INTERNSHIP REPORT

Nilesh Balu, Elaa Chaabani and Killian Babin

Mechatronic and Robotic Systems Laboratory

Ontario Tech University

$4^{th} August$ 2023

Contents

ABSTRACT

This report outlines the progress made on the project this summer. The goal at the beginning of the internship was to make the wheelchair move from an initial point to the goal position on its own. Before testing out any of the code on the wheelchair, simulations were run on Gazebo to root out obvious logical errors. Next, the focus shifted to getting the wheelchair up and running. Various sensors like stereo cameras and IMU sensors were integrated with the wheelchair. Once communication was established between the wheelchair and the user through ROS, the mapping process began. First, the wheelchair needed to be localized on 3D world coordinates. Then, gmapping was used to map the surrounding of the wheelchair. Finally, the wheelchair was made to navigate to a goal destination using P controllers while simultaneously avoiding obstacles encountered on the way.

Keywords Gazebo · GMapping · Active Monte Carlo Localization (AMCL) · ROS

1 Introduction

The goal of this project is to develop a fully autonomous electric wheelchair for use by children with motor challenges. The idea is that the user would specify what room in a building they would like to go to, and the system would automatically take them there. The system must be able to safely transport the user in the wheelchair around indoor environments such as schools and homes while avoiding both static obstacles (e.g., walls and stairs) and dynamic obstacles (e.g., people walking around). A previously developed prototype will be used to create autonomous navigation algorithms that are failsafe and easily useable by children. The navigation algorithms will be developed using the Robot Operating System (ROS).

Section 2 elaborates on the packages used in the gazebo simulation. Section 3 lists the hardware used in the wheelchair. Section 4 describes how the mapping process was carried out. Section 5 talks about the progress made in the Autonomous Navigation functionality. Finally, section 6 mentions the next steps to be taken.

2 Gazebo Simulations

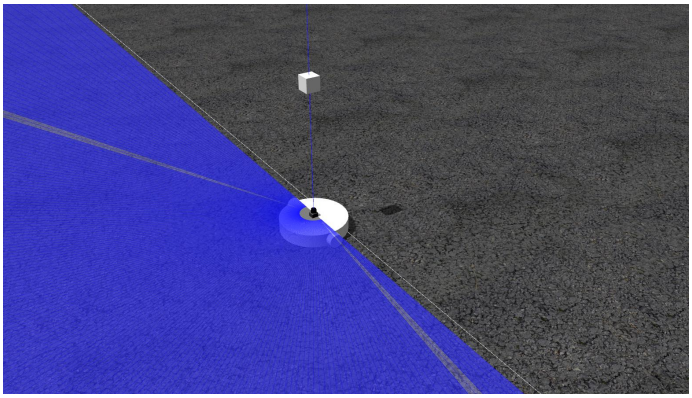

Figure 1: Robot model on Gazebo

Three packages are created for simulations on Gazebo. The *wheelchair_description* package contains a .urdf file which describes the wheelchair geometry. For the sake of simulations, the wheelchair model is just a disk with a stereo camera and a LiDAR (refer figure 1.

The *wheelchair_control* package moves the wheelchair from one position to another using a Proportional Controller. The distance between the goal coordinates and the coordinates of the current position of the wheelchair is the error that is passed into the controller. This package uses a client-server model to move the wheelchair. The *move_dist_server* moves

the wheelchair in a straight line, while the *move_distance_obstacle_avoidance_server* also avoids obstacles.

The wheelchair can map its surroundings by moving it around with your keyboard using the *teleop_twist_keyboard* package. One of two packages can be used for mapping - the *hector_mapping* package and the *gmapping* package. The *gmapping* package requires the odometry data of the wheelchair and the laser scan data, while the hector_mapping package only requires the laser scan data.

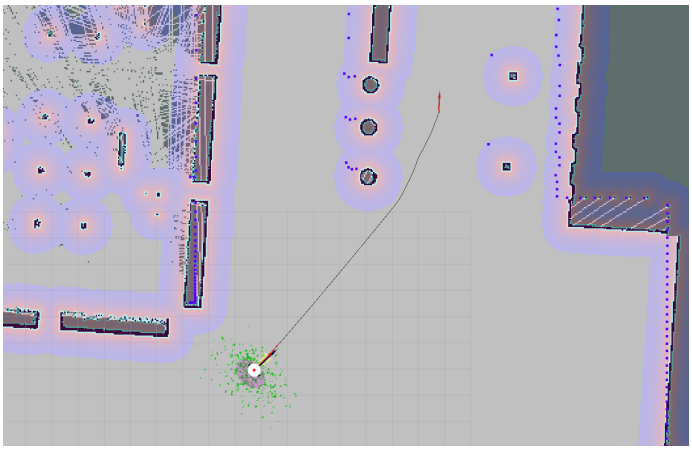

Figure 2: Autonomous Navigation of the wheelchair on Gazebo

The *wheelchair_navigation* package uses two other standard packages - the *amcl* package and the *move_base* package. Using this package, the wheelchair utilizes a map to plan its path and navigate autonomously. In *wheelchair_navigation/param*, there are six *.yaml* files which contain different parameters involved in path planning. The nature of the trajectory can be changed by tuning these values. Figure 2 shows the planned trajectory of the wheelchair.

3 Hardware

3.1 ZED camera

The ZED camera was first used as the only sensor in the wheelchair. It came with its own SDK and the *zed_wrapper* package. It was mounted on the wheelchair facing the front. The ZED camera was later replaced by two realsense d455 cameras

Figure 3: Camera mounted on the wheelchair

3.2 Intel realsense d455 cameras

The wheelchair uses two intel realsense d455 cameras, one facing the front and another facing the rear. The camera mounts are designed so that the cameras point 10 degrees downwards in each direction (see figure 3). Using two cameras increases the field of view, improving mapping and path planning. The realsense camera has its own ROS driver, and the images can be viewed using the *realsense2_camera* package.

3.3 Motor Controller - RoboteQ XDC2430

The RoboteQ motor controller comes with its own ROS driver. Using the *roboteq_diff_driver* package, the power input to the motors can be controlled.

This motor controller was found faulty because it produced a lot of noise and corrupted the encoder signals. As a result, this was replaced with the RoboClaw 2x60AHV motor controller. This controller also has its own ROS driver.

3.4 Encoders

Rotary incremental encoders were used in the wheelchair. These came inbuilt within the wheelchair. Using the encoder readings, the wheelchair can be localized in 3D world coordinates.

3.5 IMU Sensor

BNO055 IMU sensors were used in the wheelchair. The readings from this IMU can be fused with the encoder readings to get better results for the position of the wheelchair. The readings from the two sensors are passed through an Extended Kalman Filter to get the position of the wheelchair.

3.6 Arduinos

Three Arduinos were used to connect the encoders and the IMU to the laptop. Information from the sensors was communicated to the laptop using the *rosserial* package on ROS.

4 Mapping

4.1 Localization

A combination of encoders and an IMU are used to localize the wheelchair. The odometry from the encoder readings is calculated as shown:

$$\Delta x = \frac{dL+dR}{2}\ cos\theta_{avg}$$
$$\Delta y = \frac{dL+dR}{2}\ sin\theta_{avg}$$
$$\Delta\theta = sin^{-1}(\frac{dR-dL}{l})$$

where,

$$\text{l = wheelbase length}$$
$$\theta_{avg} = \frac{sin^{-1}(\frac{dR-dL}{l})}{2} + \theta$$

To get the encoder ticks on ROS, the *rosserial* package was used. The encoders are connected to an Arduino from which the encoder readings are read on ROS. The IMU readings are also obtained in the same way. The Extended Kalman Filter (EKF) is then used to estimate the wheelchair's pose better. This is done using the *robot_pose_ekf* package.

4.2 2D Mapping

The *gmapping* package was used to perform the 2D mapping. The depth image obtained from the camera is converted into a laser scan using the *depthimage_to_laserscan* package. The package does this by isolating the horizontal centre line of the depth image and converting it into a 2D laser scan. The *gmapping* packing takes this laser scan and the pose

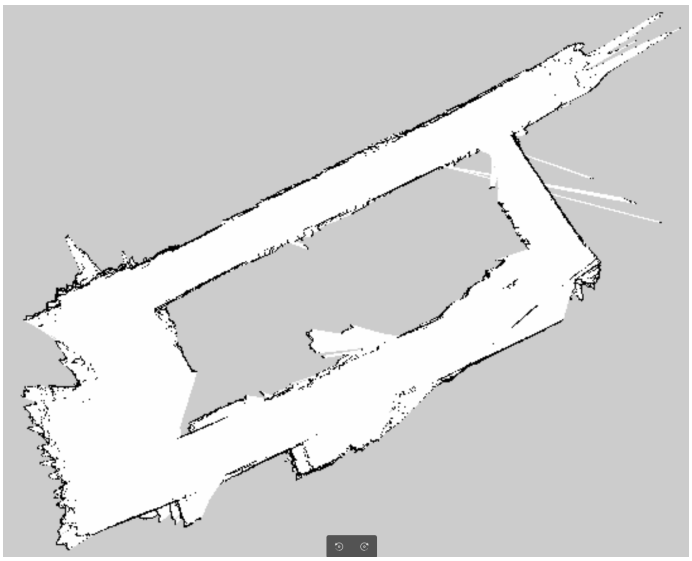

Figure 4: Map with a single camera

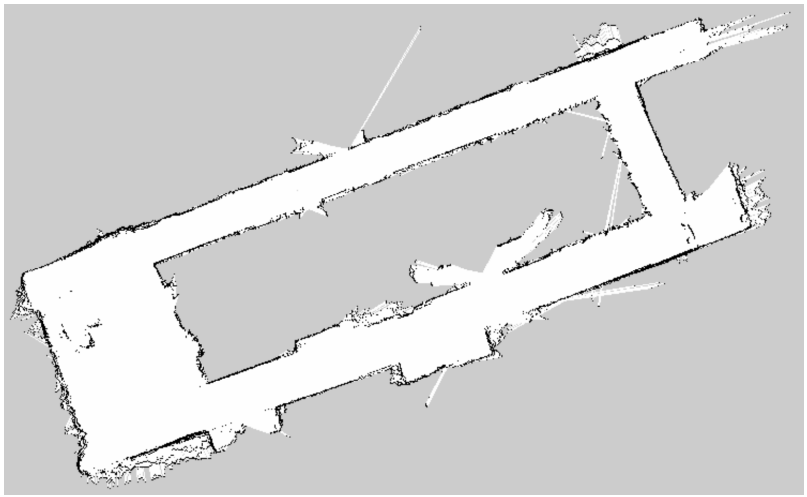

Figure 5: Map with multiple cameras

of the wheelchair from the EKF and produces a 2D map. Refer to figures 4 and 5 for the 2D maps of the hallway outside the lab.

While using two cameras, the depth image of each camera was converted into a laser scan using the *depthimage_to_laserscan* package. Then, the two laser scans were combined using the *ira_laser_tools* package to get the combined laser scan. Using static transforms, the scans were made to face opposite directions before merging.

4.3 3D Mapping

Figure 6: 3D Map of the lab (1)

Figure 7: 3D Map of the lab (2)

The RTabMap package allows for 3D mapping using just the realsense camera. Refer to figures 6 and 7 for the 3D maps of the lab. The 3D maps are saved as *.ply* files can be read and published as point clouds using the *ply_publish* package.

5 Autonomous Navigation

As of now, the wheelchair can move from the current position to a goal position in a straight line using three proportional controllers:

1. One P controller where the error is the distance between the current position of the wheelchair and the goal position.
2. Two P controllers - one for each motor, with angular velocity as the set point. This angular velocity is calculated through inverse kinematics as follows:

This is done through an action client server. In addition to moving in a straight line, the wheelchair can also avoid obstacles using the same server used in the *wheelchair_control* package.

Note: The wheelchair was mounted on wooden planks and the robot location was visualized on rviz. Since the motor controller was found faulty, this functionality could not be tested completely.

6 Future work

1. Once the RoboClaw motor controller arrives, it needs to be integrated into the wheelchair. Most of the code is ready. All instances of the RoboteQ motor controller package must be appropriately replaced with the RoboClaw package.
2. The different P controllers can be converted to PID controllers and should be tuned.
3. Slope detection and stairway detection should be implemented in 3D mapping.
4. Implement object detection for static and dynamic objects not on the map

7 Links to the code

The Catkin workspace can be found in this drive link:

The GitHub link with a well-documented readme file listing all the commands to be run can be found here: