

CHAPTER 3 – RESEARCH METHODS

In this chapter, starting from the concept of this project, the design justifications for components used as well as the reasons behind the design for mechanic, electrical and software. This project will be done in collaboration with Vinlen Aston Sidharta, a fellow batch 2017 Mechatronics student from Swiss German University. All tasks in this project are divided between two people and will all be discussed in this thesis but not as detailed for Vinlen's part of the tasks.

3.1 Conceptual Design

The main objective of this thesis is to develop a scaled-down SDC in form of an AMR that can navigate itself to reach its targeted goal for last-mile delivery. In comparison with an SDC, this AMR is expected to be in the automation level 4 driving in which the AMR will be able to locate itself in an outdoor environment and navigate to a certain targeted point then return to its initial position.

Autoware is considered the world's first "all-in-one open-source software for SDC. Autoware.Auto as the successor of Autoware.AI, is based on ROS2 with several advantages compared to its predecessors, such as adapted to modern software engineering with detailed code and continuous integration testing, improved system architecture including its interface design as well as reproducibility and stability in its library and system. Currently, Autoware.Auto targets its SDC for two cases, Autonomous Valet Parking and Autonomous Cargo Delivery. In January 2021, the valet parking functions of Autoware.Auto has been successfully implemented and showcased. The cargo delivery on the other hand is still in development. (*Autoware.Auto*, no date b)

This project concept is taken from Autoware.Auto. Driving from a certain point to a targeted goal is similar to its cargo delivery functions. However, creating an SDC

project which requires various kinds of advanced sensors and components as well as requires huge amount of cost and time, the project is decided to be not an SDC but a scaled-down SDC in form of an AMR thus reducing the difficulty of the project.

Like an SDC, an AMR also has the same process called AD stacks (Autonomous Driving) which consists of sensing, computing, and actuation. In sensing, there are two main tasks to achieve which are localization and object detection. RTK-GNSS and robot odometry using IMU and rotary encoder will be used for localization. LiDAR and 3D camera will be used for object detection as well as an ultrasonic sensor for the emergency brake. In computing, most of the information will be processed inside ROS2 firmware and its packages as well as Linorobot firmware which will be discussed further later on. In actuation, the drive of the motor will be given through the Linorobot firmware as well.

3.2 System Block Diagram and Algorithm Flowchart

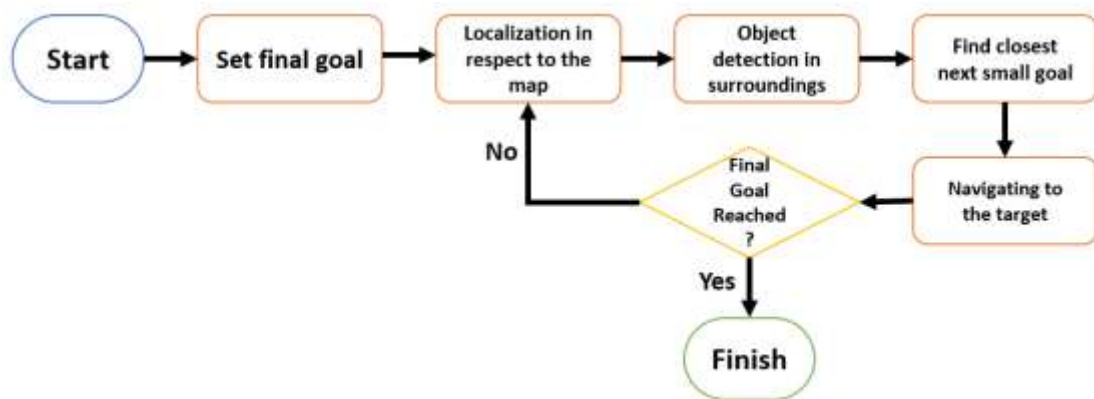


Figure 3-1 System Block Diagram and Algorithm Flowchart

Figure 3-1 shows the overall system design. The system will first request for a final goal then it will locate the position of the final goal on the map. This final goal position can be in form of latitude and longitude or x and y value. Next, using RTK-GNSS and robot odometry, the system will acquire the position of the robot through localization in the map and set it as its initial position. It will also enable its object detection functions to detect obstacles around the robot. After all sensing devices are ready and actively sensing, the system will find the closest next small goal to get to the final goal and enter its navigation algorithm to get to that particular small goal. The system will

calculate the difference in position between the robot and the final goal creating a route plan and in between this route plan is the small goal to be achieved by the robot periodically. Obstacles detected by the system's object detection will also affect how the system creates the route plan of the robot deterring its path. After completing a small goal, the robot will compare its updated position to the final goal. If it has not arrived at the designated goal, the robot will repeat the process and start over to find the next closest small goal. This process will be repeated until the robot's position is the same as the final goal.

3.3 Design Details and Components of Design

3.3.1 System Design

On an AMR or SDC system, there are three main processes which are sensing including localization and object detection, data computation, and actuation. First, localization will be used to locate the robot position in the map using data from RTK-GNSS and combined with robot odometry to set its initial position, which is visualized in Mapviz. Mapviz will subscribe to the topic from ROS2 which consists of geo-spatial position in form of latitude and longitude. This data will be shown directly on the Earth's map terrain of Mapviz as seen in Figure 3-2.



Figure 3-2 Localization Visualized on Mapviz Visualization

Second, object detection will be used to plan how the robot will move to reach its goal. Referring to the initial position of the robot, the system will calculate the route that should be taken to reach the target goal without hitting objects in the path. The robot will then know which path is safe and will plan the actuation to follow the planned path.

Third, the actuation of the robot will be done using differential drive with two motors each controlling two wheels. With differential drive, the driving of the robot will control both the left and right side independently which will result in the trajectory path depending on how much speed is given to the wheels as seen in Figure 3-3. The rotation, throttle, and shift of the robot will be controlled by how much speed on the left and right wheel.

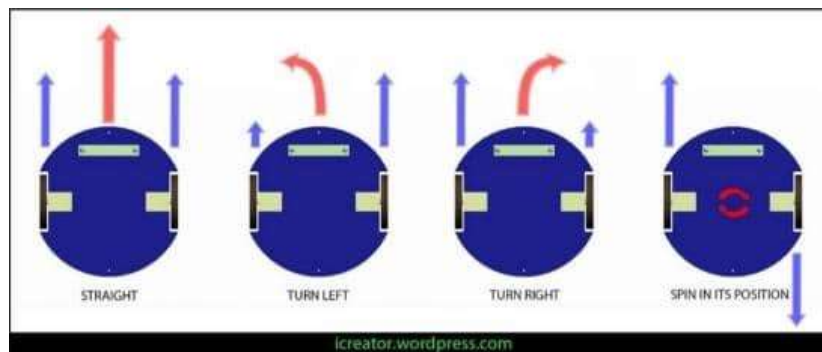


Figure 3-3 Differential Drive Actuation

When the AMR is initialized, the system will locate the position of the final goal on the map. The robot will apply its first localization using RTK-GNSS and robot odometry to obtain its initial position and create a route to reach the final goal. With 3D camera's object detection, it will detect obstacles in the route. Then, it will the next small goals to reach then actuate until it reaches the small goal. The system will then compare the current position of the robot with the final goal position. If the robot has not reached the final goal, it will repeat the process, starting from localization, object detection, small goal locating, and actuation. Once the goal has been reached, the system task is completed. The system design can be seen in Figure 3-4.

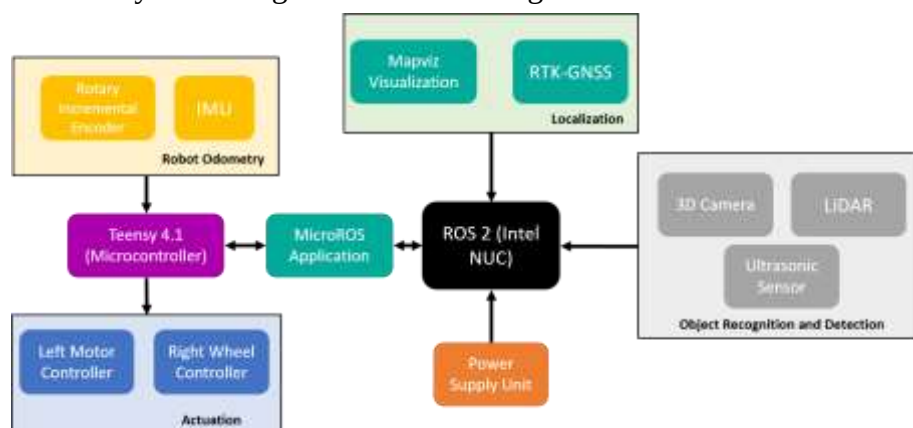


Figure 3-4 System Design

The division of the tasks for this project can be seen in Figure 3-5. Robot localization, linorobot firmware, electrical design and fabrication as well as some object detection applications will be done by Nicolas Albert Witono. Mechanical design and fabrication, and object detection applications will be done by Vinlen Aston Sidharta.

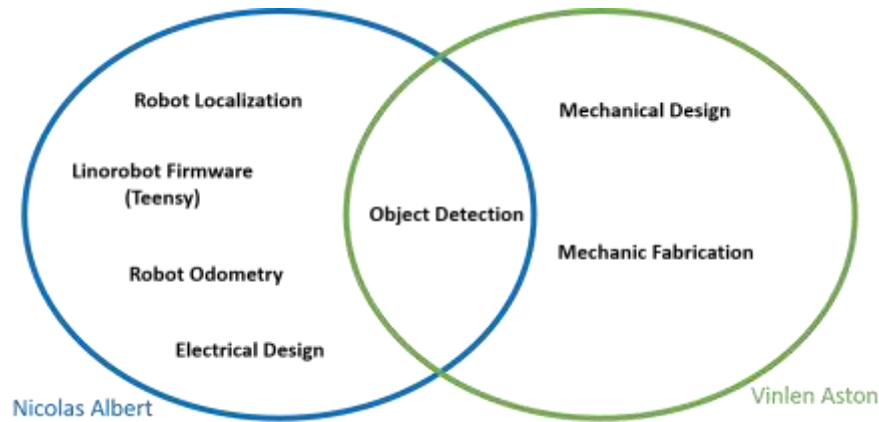


Figure 3-5 Project Tasks Division

A) Localization

Localization of a robot is more restricted outdoors as a wider and unrestricted environment is present and needs to be recognized. A system cannot depend on only one instrument for localization. By having more sensors, the position of the robot can be more accurately obtained by combining the data from these sensors. For this project, the sensors that are going to be used for localization are RTK-GNSS, IMU, and rotary encoder for robot odometry as well as Mapviz visualization.

A-1) RTK-GNSS as Main Localization

As one of the most accurate position locator, GNSS which utilizes satnav signal from satellites in real-time, will be able to give up to one to two meters accuracy of the actual location of the receiver. RTK-GNSS, which utilizes two different GNSS devices as a base station and rover, can elevate its capabilities. These two GNSS will both receive satnav signal and the base station will send out corrections periodically through RTCM connections to the rover via radio telemetry. With these corrections, the rover, which acts as the position of the robot, will have accuracy up to one to ten centimeters. There are limitations for RTK-GNSS, which is both base station and rover must have a clear view of the sky, not covered with clouds or buildings.

For this project, Sparkfun GPS-RTK2 (RTK2) module will be used which is build based on ZED-F9P module from U-Blox as seen in Figure 3-5. There will be two RTK2 modules used, one for the base station and one for the rover. It also requires two sets of antennas, one for each module as well as a USB type C connection for powering the RTK2. Figures 3-6 and 3.7 will show respectively the base station and the rover with its antenna as well as power supplied. The summarized datasheet can be seen in Table 3.1.



Figure 3-6 GPS-RTK2 Module



Figure 3-7 GPS-RTK2 as Base Station



Figure 3-8 GPS-RTK2 as Rover

Satellites Receptions	GPS, GLONASS, Galileo and BeiDou
Logic Voltage / Current	3.3 V / 68 mA – 130 mA
Time to First Fix	25s (cold), 2s (hot)
Max Navigation Rate	PVT – 25Hz; RTK – 20Hz Raw – 25 Hz
Horizontal Position Accuracy	2.5 m without RTK 0.01 m with RTK
Max Altitude	50 km
Max Velocity	500 m/s
Weight	6.8g
Dimensions	43.5 mm x 43.2 mm

*Table 3-1 GPS-RTK2 Module Specifications***A-2) Robot Odometry (IMU and Rotary Encoder)**

If RTK-GNSS gets the position of the robot from external information which is satellite signal, robot odometry will give the position of the robot from the robot itself. Robot odometry is obtained by calculating the movement of the robot which is the robot kinematics. In this case, sensors used for robot odometry are IMU and rotary encoder which gives the essential information to get the kinematics of the robot. It will record the initial position of the robot than when the robot is moving, by measuring using IMU and rotary encoder, the system can get the updated position of the robot. This is necessary for an AMR as when the RTK-GNSS signal is lost, the robot can refer to its own robot odometry to not get lost in the environment. As a self-localization method, it can also act as a reference to compare with the value from external localization such as RTK-GNSS.

For this thesis, the IMU used will be MPU9250 as seen in Figure 3-8 and the rotary encoder used will be Hayoung HE50B-8-360-3-N-24 (Shaft Type) as seen in Figure 3-9. MPU9250 consists of 3-axis accelerometers, 3-axis gyroscopes, and 3-axis magnetometers. It will be able to measure the acceleration, rotations as well as heading of the robot. HE50B will be able to measure the displacement of the robot. Altogether, the velocity and heading of the robot will be required for robot

odometry in this thesis. MPU9250 and HE50B specifications are shown in Tables 2 and 3 respectively.



Figure 3-9 MPU9250 Module



Figure 3-10 Hanyoung HE50B Incremental Rotary Encoder

Supply Voltage	3.3 – 5 V
Communication Mode	I2C / SPI
Gyroscope Type / Range	ITG3200 / 250 – 2000 deg/s
Accelerometer Type / Range	ADXL345 / 2 – 16g of force
Magnetometer Type / Range	HMC5883L / 4800 μ T
Dimension	15 mm x 25 mm

Table 3-2 MPU9250 Specification

Encoder Type	Shaft rotary incremental
No. of Pulse per Revolution	1400 pulses per revolution
Output Type	NPN Voltage Output
Voltage Range / Max Current	12-24 V / 60 mA

Weight	200g
Dimension	Ø 50 mm, axis : Ø 8 mm

Table 3-3 Hanyong HE50B Specification

A-3) Mapviz Visualization as Map Visualizer

Mapviz visualization is a visualization tool that can be used to access and visualize data required in outdoor environment tasks. As Mapviz is build based on ROS, it is compatible with various plugins, and publishing or subscribing topics are simple with these existing plugins. Plugins such as Earth's map terrain, GNSS localization, LiDAR laser scans, camera image captures, odometry, and many others will be able to ease robot navigation as Mapviz can show the condition of the robot in real-time.

B) Object Detection

Object detection is essential in SDC or AMR as they are considered the safety factor of the system. An SDC is required to be able to detect objects in the environment so it can avoid them whilst reaching its destination. Object detection can be informed of mere detecting the position of objects or recognizing the object and in SDC, object recognition is necessary. In this project, LiDAR and 3D camera will be used for object detection.

B-1) 3D Camera Depth Image to Laser Scan

With the technology of 3D cameras, the three-dimensional world can now be captured instead of two dimensions pictures using a conventional camera. From these depth images, 3D point clouds can be obtained which are defined in a coordinate system. These points can be collected to measure the distance or form the shape for object recognition. (Kadambi, Bhandari and Raskar, 2014)

For this thesis, Intel Realsense depth camera D435 as seen in Figure 3-10 will be used. It has a wide field of view and range of up to ten meters which is suitable for robotics applications. It consists of four different components which build up into a powerful depth camera, which are right imager, left imager, IR projector, and RGB module. Realsense D435 specifications are shown in Table 4.



Figure 3-11 Intel Realsense Depth Camera D435

Environment / Ideal Range	Indoor and Outdoor / 0.3-3 m
Depth Technology	Active IR Stereo
Depth Field of View	87° x 58°
Depth Output Resolution / Distance at Max Resolution	1280 x 720 / < 28cm
RGB Sensor Technology	Rolling Shutter
RGB Frame Resolution / Frame Rate	1920 x 1080 / 30 fps
RGB Sensor Field of View	69° x 42°
Dimension	90 mm x 25 mm x 25 mm

Table 3-4 Intel Realsense Depth Camera D435 Specifications

B-2) LiDAR for Object Detection

LiDAR which is capable of getting accurate distances of objects, not only excellent for localization to get position indoors but also excels in mere detecting object. Using a laser emitter and optical detector, the time of flight of the laser can be calculated to measure distances.

For this project, Slamtec's RPLiDAR A2 as seen in Figure 3-11 will be used. As a 2D LiDAR, it will be able to scan in 360° omnidirectional laser range scanning in a single plane and generate an outline map of the environment. Enhanced with the capabilities of publishing sample rates up to 8000 times, it will assist the robot in capturing objects quickly and accurately. Rated as Class 1 according to Laser Safety Standard which used low power infrared laser light emitted in a short time frame which is safe for humans. RPLiDAR A2 specifications can be seen in Table 5.



Figure 3-12 Slamtec RPLiDAR A2

Range Radius	16 m
Laser Diameter	4 cm
Sample Rate Frequency	8000 samples
Motor Drive	Brushless Motor Non-contact Drive
Scan Speed	5-15 revolutions per second
Lifespan	5 years at 24h usage
Weight	190g
Dimension	Ø 72.5 mm x 41 mm

Table 3-5 Slamtec RPLiDAR A2 Specifications

B-3) Ultrasonic Sensor for Emergency

Ultrasonic sensor which works similarly to infra-red sensors like LiDAR emits ultrasonic sound waves and receive the reflection which measures the time of flights thus obtaining distance. Despite not as quick and accurate as laser emitting sensors, ultrasonic sensors are used as proximity sensor, to detect obstacles that suddenly appears and prompt the system to execute an emergency stop.

For this project, Gravity: URM09 Ultrasonic Sensor from DFRobot.com as seen in Figure 3-12 will be used. URM09 are designed for fast ranging and obstacle avoidance application. With a built-in DAC (Digital Analog Converter), URM09 will store the measured distance as an analog and the system will be able to read the stored value from the DAC, thus reducing the delay needed for the sound wave to return. URM09 specifications can be seen in Table 6.



Figure 3-13 DFRobot.com URM09 Ultrasonic Sensor

Supply Voltage / Operating Current	3.3 – 5.5 VDC
Measurement Range	2 cm – 500 cm (adjustable)
Resolution	1 cm
Accuracy	1%
Measuring Frequency	50Hz Max
Dimension	47 mm x 22 mm

Table 3-6 DFRobot.com URM09 Ultrasonic Sensor Specifications

C) Linorobot Firmware

In an AMR system using ROS2, a computer consisting ROS2 will be required to manage the topics and nodes to control the AMR. Sensors like RTK-GNSS and LiDAR will be directly connected to the computer and publish their data into specific ROS topics. However, as the AMR uses sensors such as MPU9250, HE50B rotary encoder, and URM09 ultrasonic sensor as well as motor driver IBT2, microcontrollers are required to control all the components as they do not have independent high-level serial communication.

In *lino_base_config.h*, almost all settings and configurations of the robot can be set. Pin configurations, robot dimensions, required maximum angular velocity, robot base, wheel diameters, and PID value can all be declared in this header file. On the sensing side, there will be two sensors which are IMU MPU9250 and rotary encoder HE50B. Both IMU type and rotary encoder resolution can be set in this header file as well. Linorobot firmware has complete libraries which serve the basics of robot

applications which are libraries for encoders, IMU, robot kinematics, motor actuation, and PID calculations.

The system will first detect whether Teensy has received IMU readings in `/imu` topic. Then, it will subscribe to `/cmd_vel` topic which gives the required velocity the robot needs to drive. It will calculate through robot kinematics to obtain the required RPM of the motor to reach that required velocity. Then, it will compare the actual and required velocity to obtain the required PWM for each motor. Then, lastly, it will actuate the motor with the required PWM to achieve the required velocity of the robot. The system will finally display the current velocity of the robot in `/raw_vel` topic.

Teensy 4.1 as seen in Figure 3-13 will be the microcontroller used in this project. With its high processing capabilities, complex computations and data transfer can easily be handled. As the data obtained from sensors connected to Teensy is needed by ROS2 to process, which will then be sent back to Teensy to create a route plan, ROS2 and Teensy are required to be able to communicate with each other. Teensy 4.1 specifications can be seen in Table 7. Micro-ROS will be used to connect both ROS2 and Teensy as it is built based on ROS architecture and will create a master-slave communication between ROS2 and Teensy. Micro-ROS Arduino is an experimental micro-ROS library that is based on Arduino architecture and this library can be used to modify micro-ROS parameters.



Figure 3-14 Teensy 4.1

Processor	3.3 – 5.5 VDC
Number of Pins	2 cm – 500 cm (adjustable)
Flash Memory / RAM	1 cm
Power Output	1%
Dimension	18 mm x 61 mm x 1.57 mm

Table 3-7 Teensy 4.1 Specifications

For this project, the code that will be used is based on Linorobot firmware and software from ROS. Linorobot is an open-source ROS-compatible robot as examples seen in Figure 3-14 that is adaptable to various ROS applications including different robot bases such as 2WD (Two-wheeled Drive), 4WD, Mecanum Drive, and Ackermann Steering. Linorobot firmware is easily used and modified to fit the robot used in the project, such as IMU type, rotary encoder specs, robot design specifications, pin assignments, and many others. Since Linorobot codes are built for ROS, adaptation to ROS2 are required for it to function as desired.



Figure 3-15 Example of Linorobot

D) Intel NUC Mini PC

As an AMR not only requires a microcontroller but also a computer to compute its data, a mini PC would be ideal to put inside an AMR. Instead of putting a laptop or a PC inside an AMR that would make it look bulky, a small compact mini PC with high computation capabilities will support the AMR. For this project, Intel NUC7I7BNH-16S480 as seen in Figure 3-15 will be used. NUC7I7BNH specifications can be seen in Table 8.



Figure 3-16 Intel NUC7I7BNH

Processor	Intel Core I7-7567U (4M Cache, up to 4 GHz)
Number of Cores / Threads	2 Cores / 4 Threads
RAM	16 GB
Memory Storage	480 GB
Weight	2.3 kg

Table 3-8 Intel NUC7I7BNH Specifications

3.3.2 Mechanical Design

There are several design concepts for the design of the AMR in this project. The first design is differential drive with the use of two motors to drive one wheel each with two castor wheels. The design was created by Mechatronics batch 2014 student and after further discussions, the design was decided to be changed. The second design is differential drive but with 4WD (4-Wheel Drive) concept in which four motors are used to drive one wheel each. This design is not realizable as there are only two similar motors available to be used. The last and final design is differential drive with 4WD but two motors are used to drive two wheels each with the usage of chain for power transmission. The robot design can be seen in Figure 3-16.

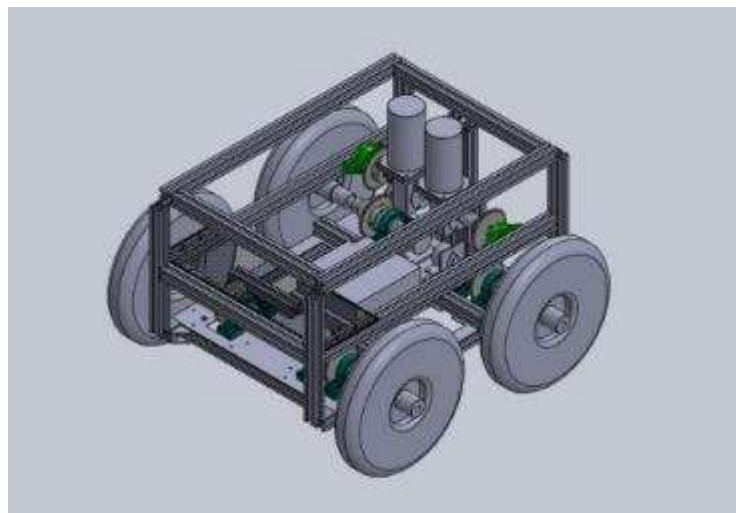


Figure 3-17 Isometric View of the Robot Design

This robot has its motor on the backside of the robot with each motor attached to a sprocket. The chain will go through the sprocket of the motor, back wheel, and front wheel thus driving both front and back wheel at the same time making it a two motors 4WD robot. The shaft and the sprocket are locked with a key and tighten with a

setscrew. Wheel-hub and motor-hub as seen in Figure 3-17 are needed to avoid welding onto the wheel or motor directly which will allow easier mounting and assembly. Both the shaft and hub will be connected with a key and tighten with a setscrew.



Figure 3-18 Wheel or Motor Hub to Shaft Design

3.3.3 Electrical and Hardware Design

In this project, a combination of both computer and microcontroller processing is required thus as seen in Figure 3-18 is the hardware design of the system. Intel NUC mini PC will be connected with Teensy 4.1, RTK-GNSS, 3D camera, and LiDAR through USB connections.

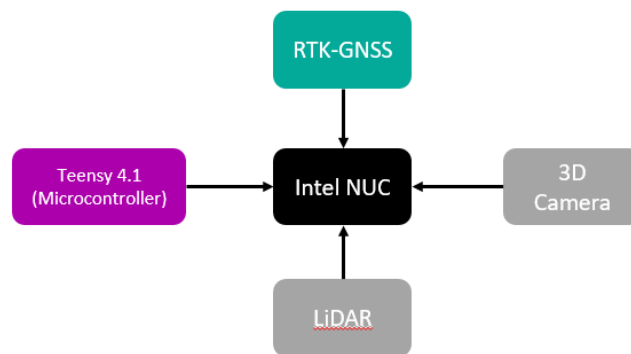


Figure 3-19 Hardware Design

Figure 3-19 will show how sensors and actuators are connected to Teensy 4.1. Electrical components such as optocoupler, resistor, capacitor, IBT2 motor driver, and regulator are used in the electrical circuit. There are three voltage sources in this system which are 12V, 5V which is obtained using regulator, and 3.3V from Teensy 4.1. Capacitors are used to stabilize all the voltage sources and resistors are used to either limit current entering the optocoupler or to use as pull-down voltage. For the rotary incremental encoder, as the encoder needs to be supplied with 12 to 24V, the output from the encoder will be also 12-24V. A voltage step-down is needed as Teensy 4.1 can only

receive input ranging from 3.3-5V and an optocoupler is used to reduce the output of the encoder. Both MPU9250 and ultrasonic sensor can directly be connected to Teensy 4.1 as 3.3-5V are their operating range. For the IBT2 motor controller, it will receive an input of 12V from the power supply unit for the motor and 5V from the regulator for the controller. The input signal to drive the motor from Teensy 4.1 will enter optocoupler which acts as an isolator to protect Teensy 4.1 if the motor generates back EMF to prevent it from damaging Teensy 4.1. All these electrical schematics are built into a PCB as seen in Figure 3-20.

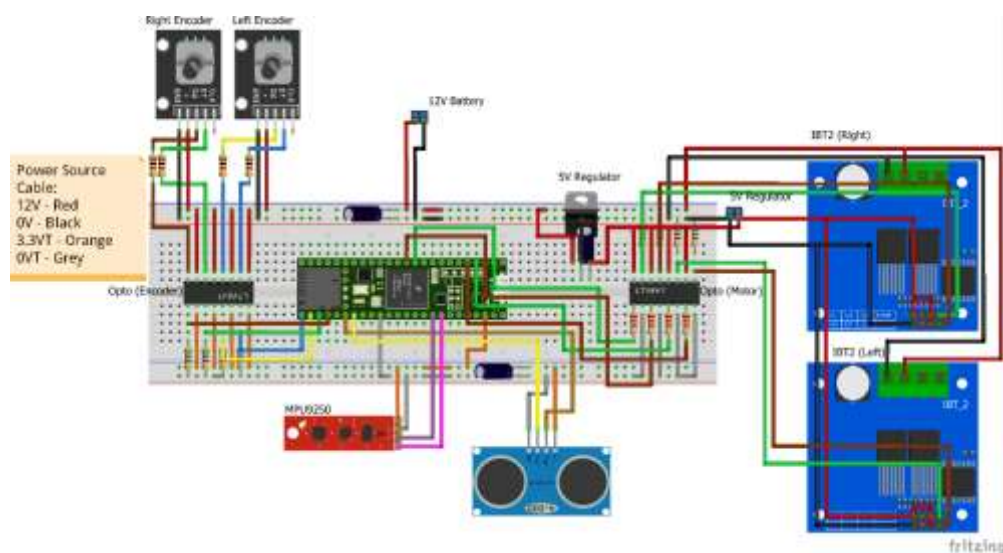


Figure 3-20 Electrical Schematic

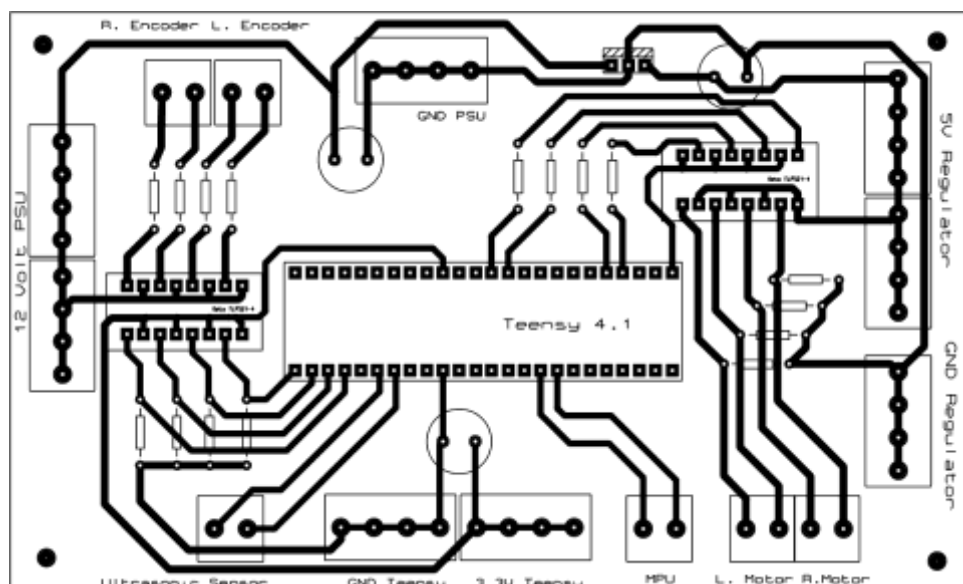


Figure 3-21 PCB Wiring Schematic

All in together, the completed electrical and hardware diagram can be seen in Figure 3-21.

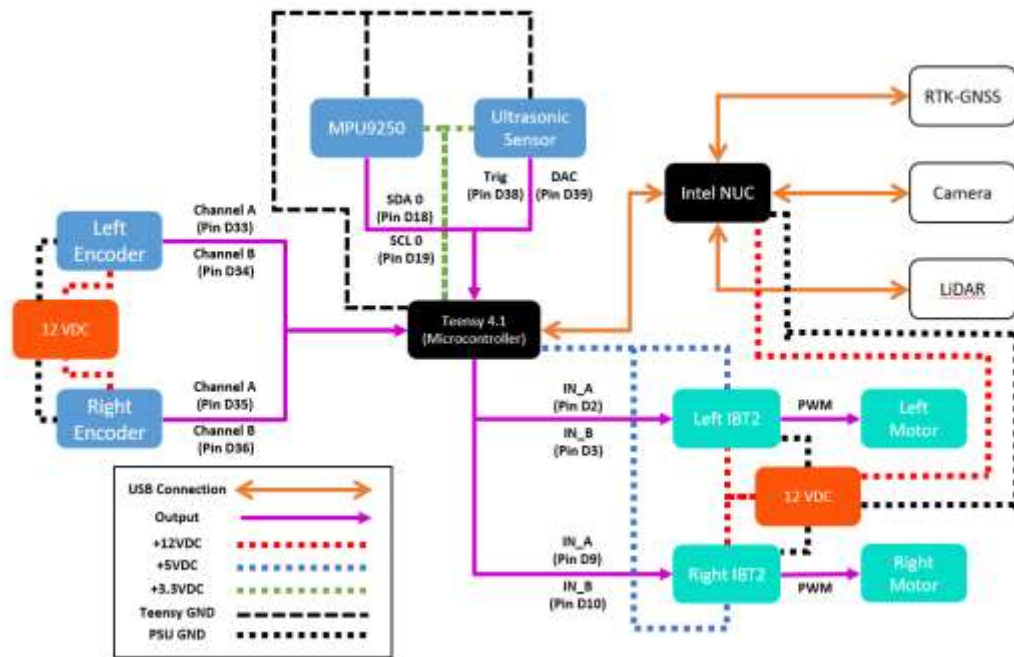


Figure 3-22 Electrical and Hardware Diagram

3.3.4 Software Design

A) U-Center by U-Blox (RTK-GNSS)

As the GPS-RTK2 module by Sparkfun is used in this thesis, the U-Center application by U-Blox can be used to set up the module into both base station and rover. As RTK2 is built based on ZED-F9P by U-Blox, U-Center is suitable for it.

To set up RTK2 as a base station, some configurations have to be changed inside U-Center. First, the RTCM3.3 communication messages for 1005, 1074, 1084, 1094, 1124, and 1230 need to be set that UART2 and USB communication are enabled as seen in Figure 3-22. Next, the port baud rate settings need to be set to 57600bps which is required by the radio telemetry from its default at 38400bps as seen in Figure 3-23. Then, as the base station needs to become a fixed position, it has to survey the weather before getting into a fixed position. In Time Mode3, set the mode into Survey-In with the minimum observation time as well as the required position accuracy as seen in Figure 3-24. Lastly, save the configuration as seen in Figure 3-25.

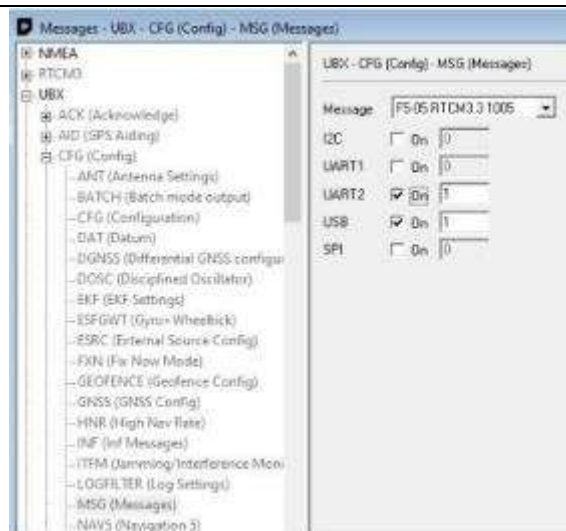


Figure 3-23 RTCM Messages Base Station Configuration

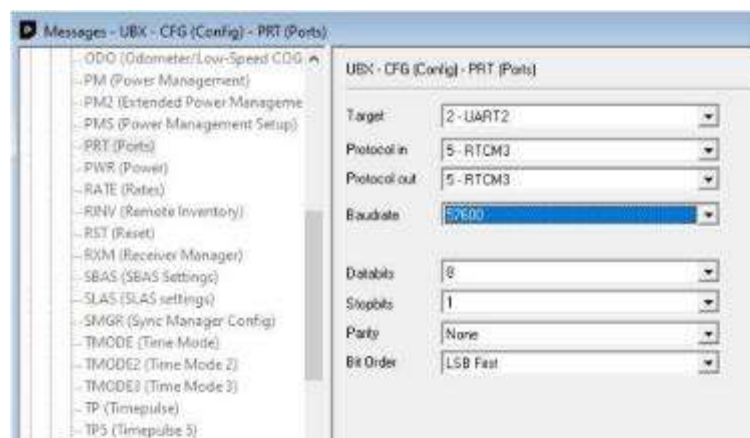


Figure 3-24 Port Base Station Configuration

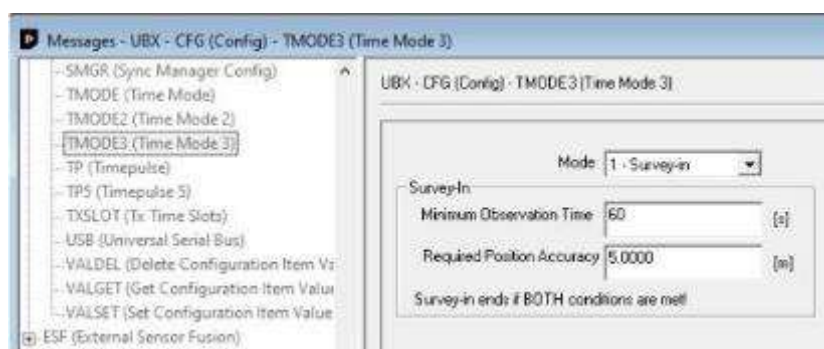


Figure 3-25 Time Mode Base Station Configuration

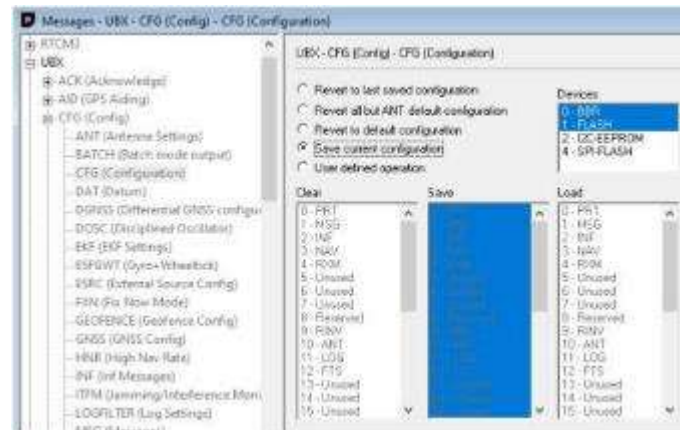


Figure 3-26 Saving Base Station Configuration

To set up RTK2 as a rover, less configuration also requires to be changed inside U-Center. The port baud rate settings need to be set to 57600bps as well like the base station. Lastly, save the configuration so it applies to every startup. Whenever the base station has become fixed mode, the rover will receive RTCM corrections using the radio telemetry and the rover will change its mode from 3D mode to RTK Float then lastly to RTK Fixed.

B) NMEA Navsat Driver Package (RTK-GNSS)

This ROS2 package parse NMEA strings and publishes simple GNSS messages in form of latitude, longitude, and altitude. It publishes `/fix` topic which is `sensor_msgs/msg/NavSatFix` type. This package can be obtained by using the following command in the terminal:

```
$ git clone -b ros2 https://github.com/ros-drivers/nmea_navsat_driver.git
```

The node can be executed by using the following command in the terminal:

```
$ ros2 run nmea_navsat_driver nmea_serial_driver
```

C) Slamtec RPLiDAR Package (LiDAR)

This ROS2 package is specially used for RPLiDAR devices which are 2D laser scanner RPLiDAR A1, A2, and A3. This package publishes `/scan` topic which is `sensor_msgs/msg/LaserScan` type. This package can be obtained by using the following command in the terminal:

```
$ git clone -b ros2 https://github.com/Slamtec/rplidar_ros.git
```

The node can be executed by using the following command in the terminal:

```
$ ros2 run rplidar_ros rplidar_node
```

D) Turtlebot3 Package for Teleoperation

Turtlebot3 is a ROS standard platform robot whose main core functions are SLAM (Simultaneous Localization and Mapping), Navigation, and Mapping. The package also enables the robot to be driven remotely using teleoperation. Teleoperation node will publish `/cmd_vel` topic which is `geometry_msgs/msg/Twist` type. This package can be obtained by using the following command in the terminal:

```
$ git clone -b foxy-devel https://github.com/ROBOTIS-GIT/turtlebot3.git
```

The node can be executed by using the following command in the terminal:

```
$ ros2 run turtlebot3_teleop teleop_keyboard
```

E) Micro-ROS Agent

This ROS2 package will enable ROS2 architectures to be able to communicate with Teensy 4.1 via serial communications so topics can be published or subscribed by both sides. This package can be obtained by using the following command in the terminal:

```
$ git clone -b foxy https://github.com/micro-ROS/micro_ros_setup.git
```

Micro-ROS agent can be created by using the following command in the terminal:

```
$ ros2 run micro_ros_setup create_agent_ws.sh  
$ ros2 run micro_ros_setup build_agent.sh
```

Finally, Micro-ROS application can be started by using the following command in the terminal:

```
$ ros2 run micro_ros_agent micro_ros_agent serial --dev [device]
```

F) Micro-ROS Arduino

Micro-ROS Arduino will enable to code ROS2 codes on Arduino IDE so that Teensy 4.1 and ROS2 can communicate with each other. Using Micro-ROS Arduino, topics and nodes can be created to be used by ROS2 as well. This package can be obtained by using the following command in the terminal:

```
$ git clone -b foxy https://github.com/micro-ROS/micro_ros_arduino.git
```

Inside Arduino IDE, Micro-ROS Arduino libraries can be included directly and can be directly used.

G) Mapviz Visualization Package

This package will enable visualization of ROS topics such as GNSS fix, LiDAR laser scan, odometry, and many more. This package can be obtained by using the following command in the terminal:

```
$ git clone -b dashing-devel https://github.com/swri-robotics/mapviz.git
```

The package can be launched by using the following command in the terminal:

```
$ ros2 launch mapviz mapviz.launch.xml
```

H) RSasaki EKF Localization Package

This package will combine robot odometry `/odom`, IMU data `/raw_imu` as well as GNSS data `/fix`. The output from the package will publish `/current_pose` topic which is `geometry_msgs/msg/PoseStamped` type. This package can be obtained by using the following command in the terminal:

```
$ git clone -b master https://github.com/rsasaki0109/kalman_filter_localization.git
```

I) Linorobot Package (ROS1)

This package will be used to control the motor with a PID controller, read rotary encoder data, read IMU data, and create related required ROS topics through Teensy 4.1. `/cmd_vel` topic of `geometry_msgs/msg/Twist` type is used to subscribe the required velocity the robot needs to move. `/raw_vel` topic of `geometry_msgs/msg/Twist` type is used to publish the velocity readings from the rotary encoder. `/raw_imu` topic of `sensor_msgs/msg/Imu` type is used to publish the measured data of IMU. This package can be obtained by using the following command in the terminal:

```
$ git clone -b master https://github.com/linorobot/linorobot.git
```

J) GNSS Geospatial Position to XYZ Coordinates

This package will be used to convert GNSS geospatial position which consists of latitude, longitude, and altitude in `/fix` topic of `sensor_msgs/msg/NavSatFix` type to XYZ coordinates in `/gps_pose` topic of `geometry_msgs/msg/PoseStamped` type. For this project, XYZ coordinates will be on the ENU (East, North, Up) coordinates. This package can be obtained by using the following command in the terminal:

```
$ git clone https://github.com/ShantihBlah/ROS2-GPS2xyz.git
```

Inside the package, there are two nodes to be executed, which are first setting up the reference GNSS position received from the GNSS which can be started by using the following command in the terminal:

```
$ ros2 run geodetic_utils set_gps_reference_node
```

The other package is to convert the GNSS position into XYZ coordinates using the referenced position given from *set_gps_reference_node* in */gps_init* topic of *geometry_msgs/msg/Vector3* type. The node can be started by using the following command in the terminal:

```
$ ros2 run geodetic_utils gps_to_pose_conversion_node
```

K) Intel RealSense Package for ROS2

This package is used to enable the usage of RealSense cameras by Intel on Ubuntu with ROS2. It will enable the drivers for RealSense and allows the usage of the RealSense Library. The package can be obtained by using the following command in the terminal:

```
$ sudo apt-get install ros-foxy-realsense2-camera
```

L) Depth Image to Laser Scan Package for RealSense

This package is used to convert depth image from RealSense which is on */depth* topic of *sensor_msgs/msg/Image* type into laser scan data which is on */scan* topic of *sensor_msgs/msg/LaserScan* type. This package can be obtained by using the following command in the terminal:

```
$ git clone -b foxy-devel https://github.com/ros-perception/depthimage_to_laserscan.git
```

3.4 Design Calculation and Modeling

3.4.1 Robot Kinematics

Robot kinematics determine the motion of the robot and for this project, 4WD differential drive will be used which uses the rotational speed of each wheel to determine the robot kinematics. The robot will have four wheels with two motors that drive two wheels each separating it into the left motor and right motor. This will result in the velocity of the front left wheel equal to the back left wheel and the velocity of the front right wheel equal to the back right wheel. The model to calculate the robot kinematics is as shown in Figure 3-26.

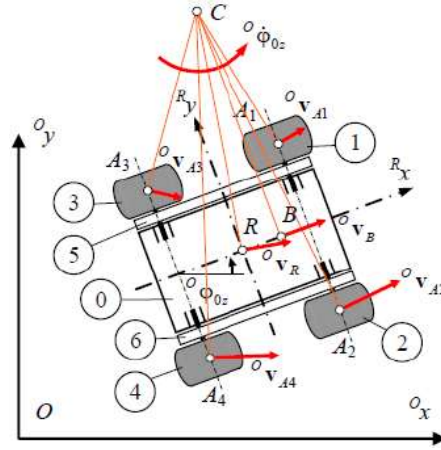


Figure 3-27 Robot Kinematics Model ($A1A3 = A2A4 = L$,
 $A1A2 = A3A4 = W$)

The robot motion is visualized in Oxy plane of the fixed coordinate system $\{O\}$. The moving coordinate system which denotes the robot center position is $\{R\}$. The position and orientation of the robot can be described by the vector of generalized coordinates:

$${}^O\mathbf{q} = [{}^OX_R, {}^OY_R, {}^O\varphi_{0z}]^T$$

${}^OX_R$, ${}^OY_R$ are coordinates of point R of the robot and ${}^O\varphi_{0z}$ denotes the angle of rotation of the robot about Oz axis with respect to the fixed coordinate system $\{O\}$. Vectors of generalized velocities respectively in $\{O\}$ and $\{R\}$ coordinate systems can be written as:

$${}^O\mathbf{q}' = [{}^OX'_R, {}^OY'_R, {}^O\varphi'_{0z}]^T, {}^R\mathbf{q}' = [{}^RX'^O, {}^RY'^O, {}^R\varphi'^O_{0z}]^T$$

where: ${}^O V_{Rx} = {}^O X'_R$, ${}^O V_{Ry} = {}^O Y'_R$, ${}^R V^O_{Rx} = {}^R X'^O_R$, ${}^R V^O_{Ry} = {}^R Y'^O_R$

These two vectors satisfy the relationship:

$${}^O \mathbf{q}' = {}^O \mathbf{J}^R {}^R \mathbf{q}' ,$$

Where matrix ${}^O \mathbf{J}^R$ has the following form:

$${}^O \mathbf{J}^R = \begin{bmatrix} \cos({}^O \varphi_{0z}) & -\sin({}^O \varphi_{0z}) & 0 \\ \sin({}^O \varphi_{0z}) & \cos({}^O \varphi_{0z}) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

With assumption of ${}^R V^O_{Rx} = 0$, vector of generalized velocities ${}^O \mathbf{q}'$ can be defined based on kinematic equations of motion in the form:

$${}^O \mathbf{q}' = \begin{bmatrix} {}^O X'_R \\ {}^O Y'_R \\ {}^O \varphi'_{0z} \end{bmatrix} = \begin{bmatrix} \cos({}^O \varphi_{0z}) & 0 \\ \sin({}^O \varphi_{0z}) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} {}^R V^O_{Rx} \\ {}^R \varphi'^O_{0z} \end{bmatrix},$$

Where vector ${}^R \mathbf{v} = [{}^R V^O_{Rx} \quad {}^R \varphi'^O_{0z}]^T$ contains respectively component of velocity of the point R of the robot on the Rx direction of $\{R\}$ coordinate system and the yaw rate of the robot.

The above equation is valid in case when the robot moves on horizontal ground. Point C in Figure XX is the instantaneous center of rotation of the robot frame. Projection of this point on Rx axis is indicated with point B . The analyzed robot is the nonholonomic system in which the robot is in a continuous closed circuit of its parameters. From the distribution of velocity vector of point B , it follows that it is subjected to nonholonomic constraints of the form:

$$\mathbf{a} {}^O \mathbf{q}' = 0 ,$$

where $\mathbf{a} = [-\sin({}^O \varphi_{0z}), \cos({}^O \varphi_{0z}), {}^R X_C]$ is the vector of nonholonomic constraints. In turn, projections of velocities of points A_i on Rx and Ry axes of the $\{R\}$ coordinate system satisfy the relationships

$$\begin{aligned} {}^R V^O_{lx} &= {}^R V^O_{A1x} = {}^R V^O_{A3x}, & {}^R V^O_{rx} &= {}^R V^O_{A2x} = {}^R V^O_{A4x}, \\ {}^R V^O_{fy} &= {}^R V^O_{A1y} = {}^R V^O_{A2y}, & {}^R V^O_{by} &= {}^R V^O_{A3y} = {}^R V^O_{A4y}, \end{aligned}$$

where: l – wheels of the left-hand side ($l = \{1,3\}$), r – wheels of the right-hand side ($r = \{2,4\}$), f – front wheels ($f = \{1,2\}$), b – back wheels ($b = \{3,4\}$).

In the case of the plane motion of the robot, the velocity of the vector of the point R depends on the angular velocity ${}^R\varphi'_{0z}$ and radius of curvature R_z of the path according to the formula:

$${}^RV^O_R = {}^R\varphi'_{0z} \mathbf{R}_z,$$

where ${}^RV^O_R = {}^R\varphi'_{0z} \mathbf{R}_z$ is the velocity with which the point R moves with respect to the stationary coordinate system $\{O\}$.

In addition, in a general case the acceleration vector ${}^R\mathbf{a}^O_R$ of the point R , expressed in the robot coordinate system $\{R\}$, has both tangential and normal components, that is, the following dependencies are valid:

$${}^R\mathbf{a}^O_R = {}^R\mathbf{a}^O_{R\tau} + {}^R\mathbf{a}^O_{Rn}$$

where ${}^R\mathbf{a}^O_{R\tau}$ and ${}^R\mathbf{a}^O_{Rn}$ denote respectively vectors of tangential and normal accelerations of the point R .

To determine velocity components of the point R , based on the known acceleration vector ${}^R\mathbf{a}^O_R$, the following equations should be used:

$${}^RV'^O_{Rx} = {}^Ra^O_{Rx} + {}^RV^O_{Ry} {}^R\varphi'_{0z}, \quad {}^RV'^O_{Ry} = {}^Ra^O_{Ry} + {}^RV^O_{Rx} {}^R\varphi'_{0z}$$

Those equations result from projecting acceleration vectors on axes of $\{R\}$ coordinate system.

Projections of velocities of points A_i on R_x and R_y axes of the coordinate system $\{R\}$ can be written as functions of generalized velocities in the form:

$$\begin{bmatrix} {}^RV^O_{lx} \\ {}^RV^O_{rx} \\ {}^RV^O_{fy} \\ {}^RV^O_{by} \end{bmatrix} = \begin{bmatrix} 1 & -W/2 \\ 1 & W/2 \\ 0 & L/2 - {}^RX_C \\ 0 & -L/2 - {}^RX_C \end{bmatrix} \begin{bmatrix} {}^RV^O_{Rx} \\ {}^R\varphi'_{0z} \end{bmatrix} = \begin{bmatrix} 0 & -W/2 + {}^RY_C \\ 0 & W/2 + {}^RY \\ 1 & L/2 \\ 1 & -L/2 \end{bmatrix} \begin{bmatrix} {}^RV^O_{Ry} \\ {}^R\varphi'_{0z} \end{bmatrix}$$

Moreover, projections of the velocity of the point A_i , which belongs to the i -th wheel, on R_x and R_y axes of $\{R\}$ coordinate system depend on the angular velocity of spin of the wheel θ'_i and the velocity of slip, that is, the following equations are satisfied:

$${}^RV^O_{Aix} = \theta'_i r_i + {}^RV^O_{Six}, \quad {}^RV^O_{Aiy} = {}^RV^O_{Siy},$$

where ${}^R V_{Six}^O$, ${}^R V_{Siy}^O$ are components of slip velocity in $\{R\}$ coordinate system, that is, the velocity of motion of points of the wheel which are in contact with the ground with respect to the ground.

Thus, the generalized velocities can be written in the form:

$${}^R V_{Rx}^O = \frac{({}^R V_{lx}^O + {}^R V_{rx}^O)}{2}, {}^R V_{Ry}^O = {}^R \varphi'_{Oz} {}^R X_B,$$

$${}^O \varphi'_{Oz} = \frac{({}^R V_{rx}^O - {}^R V_{lx}^O)}{W} = \frac{({}^R V_{by}^O - {}^R V_{fy}^O)}{L},$$

Coordinates of the instantaneous center of rotation C can be calculated from dependencies:

$${}^R X_C = {}^R X_B = -\frac{{}^R V_{Ry}^O}{{}^R \varphi'_{Oz}}, {}^R Y_C = \frac{{}^R V_{Rx}^O}{{}^R \varphi'_{Oz}}$$

The position of point B has a critical influence on the stability of robot motion. If the origin of the coordinate system associated with the robot is situated in the middle of the distance between the front and rear wheels, then one may assume that coordinates of the point B should satisfy the relationship:

$${}^R X_B \in (-L/2, L/2).$$

3.4.2 Motor Sizing

As the actuation of the robot, the motor sizing calculations of the robot is as following:

Wheel Diameter = 33 cm

Mass of the robot = 35 kg

No. of Motors = 2 (Each drives two wheels)

Required Velocity = 0.5 m/s

Required Acceleration = 0.25 m/s²

The formula for rolling coefficient :

$$Crr = 0.005 + \left(\frac{1}{p}\right)(0.01 + 0.0095 \left(\frac{v}{100}\right)^2)$$

c = rolling coefficient

p = tire pressure (bar)

v = velocity (km/h)

The formula for Torque

$$\tau = F * r$$

$F = \text{force (N)}$

$R = \text{radius (m)}$

The formula for Angular Velocity

$$w = \frac{v}{r}$$

$v = \text{velocity (m/s)}$

$r = \text{radius (m)}$

From the robot specifications, thus the rolling coefficient is:

$$C_{rr} = 0.005 + \left(\frac{1}{2.0684}\right) \left(0.01 + 0.0095 \left(\frac{1.8}{100}\right)^2\right) = \mathbf{0.0098}$$

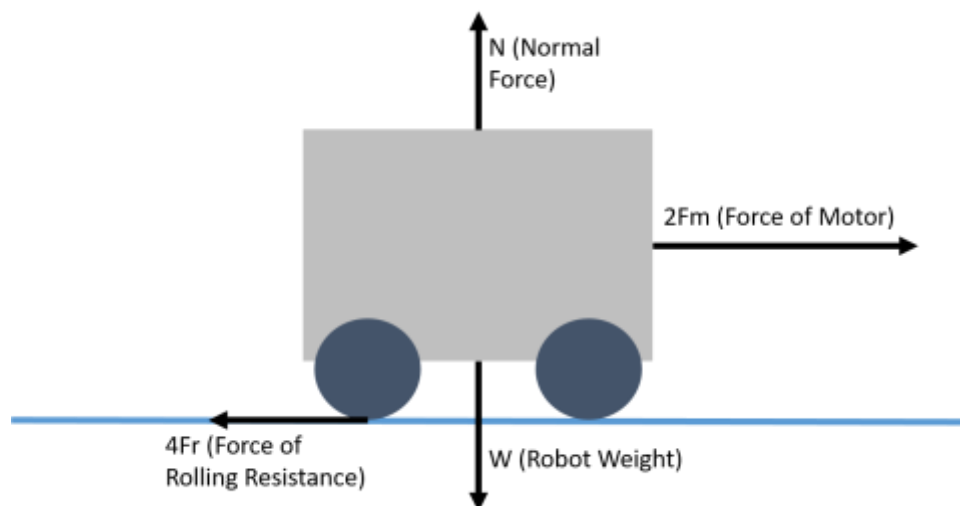


Figure 3-28 Free-Body Diagram of Robot Movement

Referring to Figure 3-27 as the free-body diagram of the robot, thus the forces on the robot can be calculated as follow:

Then look for Sigma Fy to obtain the normal force that is applied on the robot.

$$\Sigma F_y = 0 \rightarrow N - W = 0$$

$$N = 35 * 9.81 = \mathbf{343.35 \text{ N}}$$

$$Fr = \frac{n}{4} * Crr = \frac{343.35}{4} * 0.0098 = \mathbf{0.842 \text{ N}}$$

Then Sigma Fx needs to be calculated to obtain the required force of the motor.

$$\Sigma Fx = 0 \rightarrow 2Fm - 4Fr = m * a$$

$$2Fm = 35 * 0.25 + 4 (0.842) = \frac{12.118}{2} = \mathbf{6.06 \text{ N}}$$

After obtaining the force of the motor, calculate the torque required and the angular velocity.

$$\tau = Fm * rw = 6.06 * 0.165 = \mathbf{0.9999 \text{ Nm}}$$

$$w = \frac{v}{r} = \frac{0.5}{0.165} = \mathbf{3.03 \text{ rad/s}}$$

Using torque and angular velocity, the power rating of the motor can be obtained.

$$P = \tau * w = 0.9999 * 3.03 = \mathbf{3.03 \text{ Watt}}$$

Multiplied with safety factor (1.5) and divided with transmission loss and motor efficiency

$$P_{req} = \frac{P}{\text{Transfer Loss} * \text{Motor Efficiency}} * SF = \frac{3.03}{0.7 * 0.8} * 1.5 = \mathbf{8.116 \text{ Watt}}$$

Thus, from the motor sizing, to drive the robot on flat surfaces, **8.116 Watt** is required.

3.5 Prototype Fabrication

Figure 3-28 shows the prototype of the robot with the robot frames, motors, power transmissions, and wheels. In Figure 3-29, it is shown on the PCB wirings for Teensy part of the components.



Figure 3-29 Robot Prototype

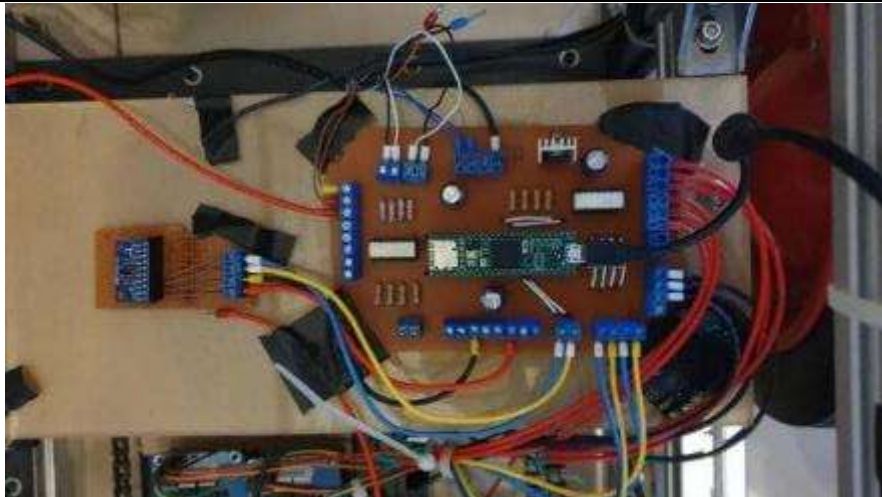


Figure 3-30 Electrical Wiring on PCB