DEVELOPMENT OF AN UNMANNED GROUND VEHICLE PHOTOGRAMMETRY DEVICE

Muammar Royyan Ibrahim

Supervisor:
Zair Asrar bin Ahmad

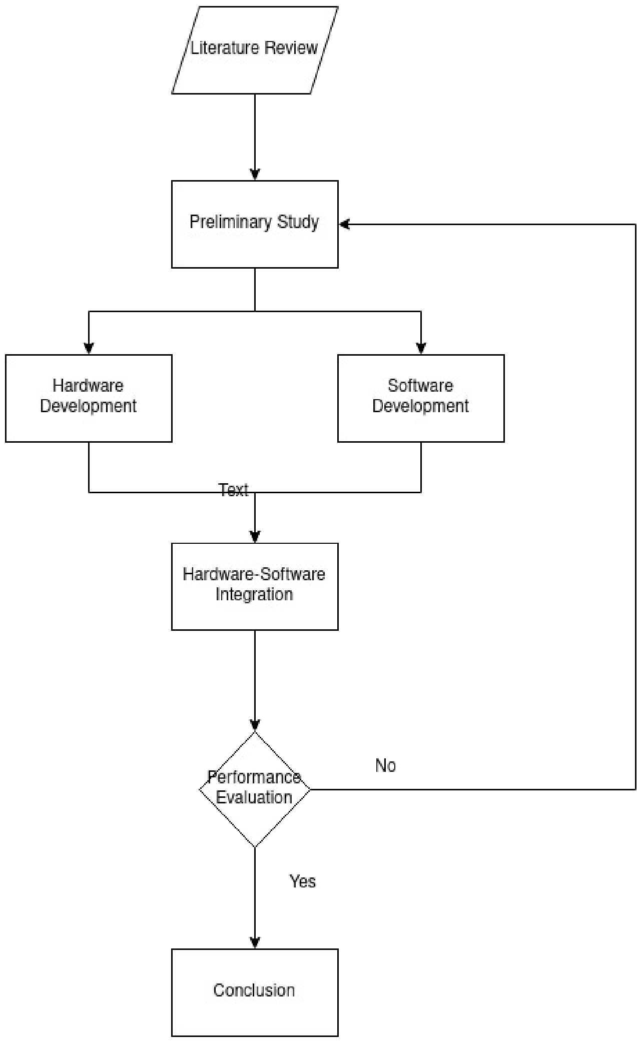
Faculty of Mechanical Engineering
Faculty of Engineering
2022

CHAPTER 3: RESEARCH METHODOLOGY

3.1 Introduction

This chapter describes the methodology of this project. The four tasks are Photogrammetry experimentation, software development, hardware fabrication and functionality testing. Figure 3.1 shows the flow chart of research methodology for this project.

Figure 3.1 Research methodology



3.2 Preliminary Study and design

The purpose of this preliminary study is to evaluate the quality of models that can be produced using traditional turntable methods. Several camera resolutions will be tested to determine the camera sensor quality required by UGV to produce image samples fit for 3D reconstruction. For this experiment a turntable was produced with an object inside, and a tripod was used to keep the camera steady, the test was conducted indoors with room lighting. Figure 3.3 represents the setup used for the experiment. The tests will include; (1) Low resolution, high sample size model, (2) High resolution, medium sample size model, (3) High resolution, long distance model, medium sample size. Figure 3.2 summarises the experiments to be taken.

Experiment	Camera quality	Sample size (n = photos)	Distance from object	Notes	Purpose
1	2mp	70	20cm	-	Determine if ov2640 can perform photogrammetry
2	16mp	36	20cm	-	Assumed target camera quality
3	16mp	36	50cm	-	Determine maximum radius the UGV can scan

Figure 3.2 Experiment list

Figure 3.3 Turntable setup used

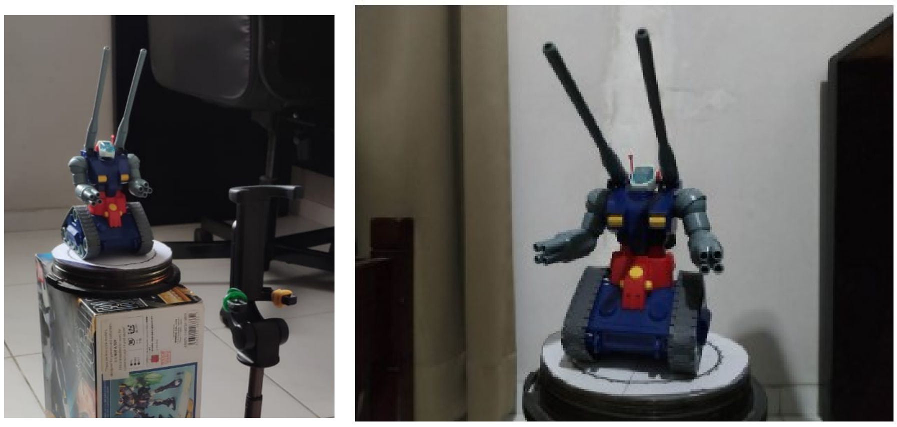


Figure 3.4 Results from experiment with 2mp camera quality & high sample size

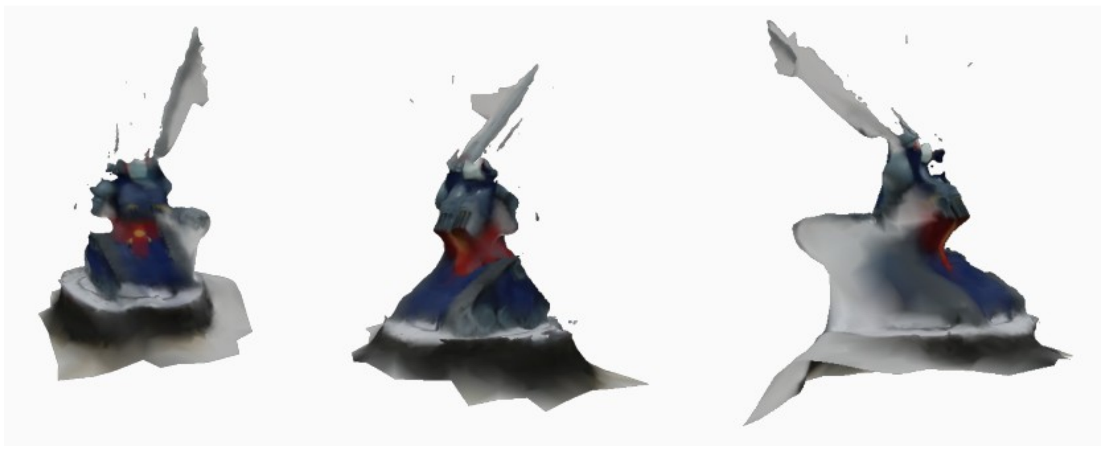
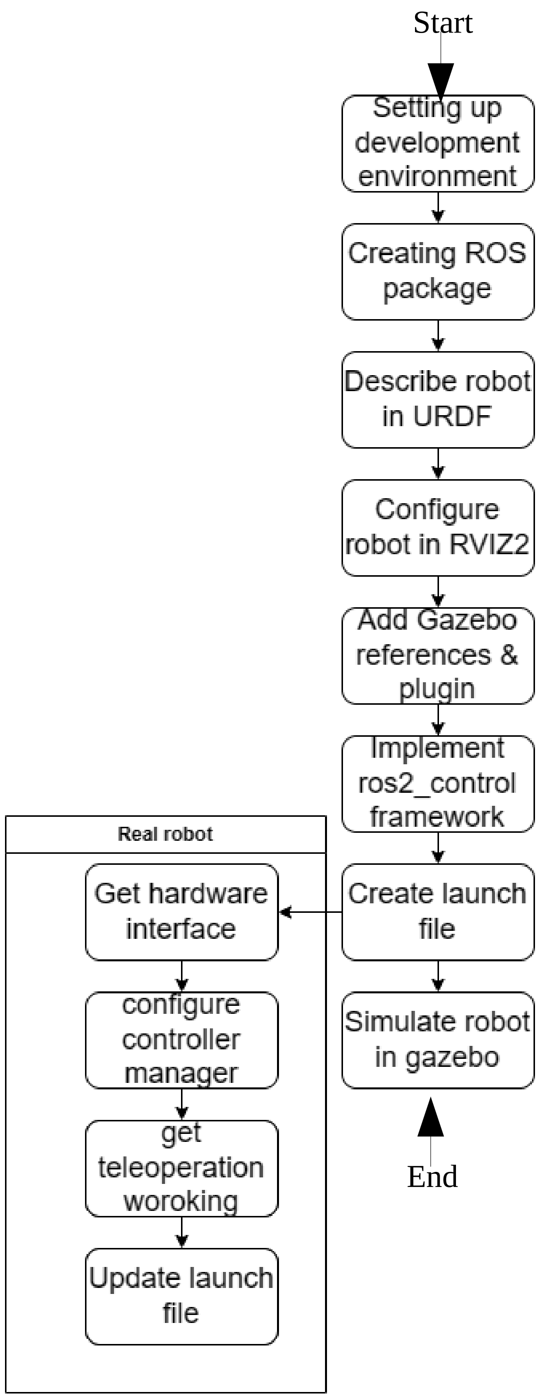


Figure 3.4 depicts the 3D model produced using low camera quality, it is similar to the table model as a result of rover mobile robot from literature review [3], the model is incomplete and textures are poor. Due to this result it is concluded to use high resolution 16mp webcam for the mobile robot solution.

3.3 Software Development

This section covers the process to develop software for the UGV using Robot operating system ROS2, it covers first the simulation aspect and modification of program for the real robot. Figure 3.5 is a flowchart for software development methodology.

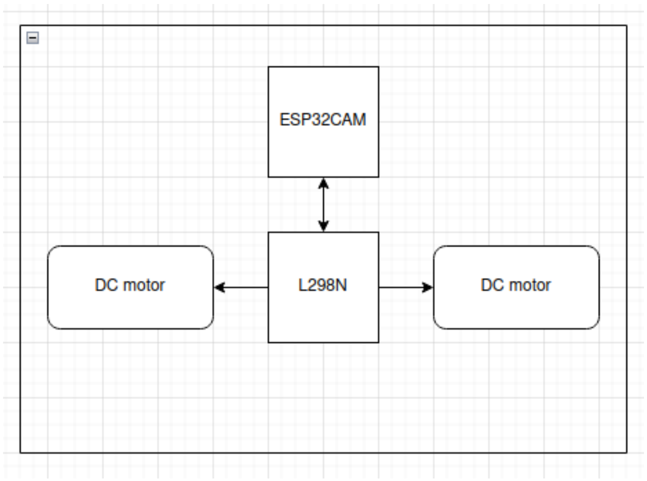
Figure 3.5 Flowchart of softrware methodology



3.3.1 Preface

As concluded from the literature review, a differential drive robot (2 drivers, 1 caster), is favourable for this application when compared to mechanuum wheel robot, and 2wd (2 wheel drive) robot with stearing micro. There are many types of differential drive robots, such as tracekd or 6 wheeled robots that utilize 2 driving motors, these setup are more complex and suitable for roguh environment, however the robot will lose a degree of freedom provided by the caster wheel in terms of motion, as well as higher fabrication cost and development time. Another advanrtage of differential drive robots is its ease in programming as it only requires two parameters to be modified; input of x direction velocity and z angular velocity.

The proposed solutions for a differential drive robot that takes pictures based on literature review is an esp32cam based robot or a raspberry pi controlled mobile robot. Figure 3.6 shows the control chart for these solutions. In both these robots software represents the control aspect of the robot, for esp32cam diffrential drive robot it would require code written in C++ to be uploaded via arduino IDE, the arduino compatible sketch will need to include a main '.ino' file, '.h' headers and a webserver in input instructions. The esp32 microcontroller directly controls an L298N motor driver to provide target velocities to dc motor.



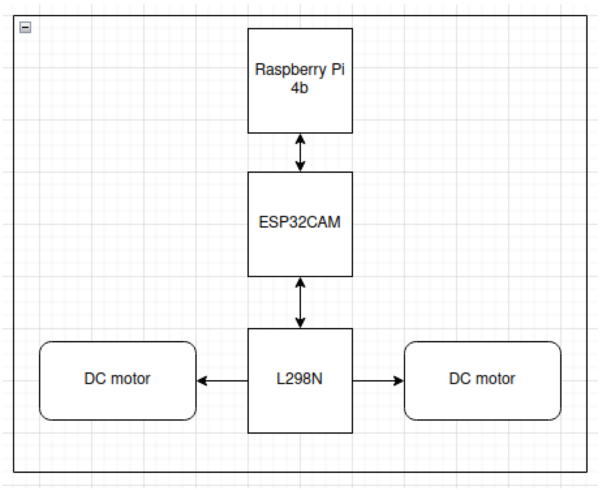


Figure 3.6 Control chart for ESP32cam robot (left) and Raspberry based robot (right)

Alternatively, a raspberry pi controlled mobile robot is another solution, this setup holds the advantage of using more advanced controllers such as ROS (Robot operating system). The ROS compatible mobile robot solution was chosen for this study for several reasons, first the esp32cam uses an ov2640 camera, which is an soc (system on chip) with integrated camera. This device is suitable for IoT purposes as it uses a 2MP sensor with 1600x1200 UGXA array while maintaining a low cost. However Raspberry Pi can use any USB compatible camera, from cheap webcams to more expensive dashcams. Secondly, a ROS based robot can be controlled more securely through SSH network between development computer and Raspberry Pi. Lastly a ROS based mobile robot solution is simply more modular for future expansions, for example more a LiDAR sensor can be added to use sensor fusion, or swarm robotics algorithm can be implemented. In this study, the Raspberry Pi running ROS2 acts a robot controller, it will be connected to an Arduino Uno acting as a motor controller, L298N as the motor driver and 2 DC motors. The software required would be a ROS2 package that output motor speeds to the Arduino which then sends it to the motor driver. Altough the Raspberry can also act as a motor controller, seperating the 2 systems reduces the resource draw of the Pi, and keeps the sytem modular which helps during the troubleshooting process.

3.3.2 Setting development environment

Robot operting system (ROS) is an opensource compilation of liobraries developed by open robotics. It is a software which allows a communication layer between all the parts of the robot for its normal functioning. In this study we use ROS2, an upgrade of the original ROS software, there are significant improvements over ROS, such as python3 suppoort and DDS support, which allows for realtime data exchange. The specific ROS2 distro used is ROS2 foxy. This software version is currently only available for Ubuntu 20.04.

Gazebo is a 3D dynamic simulator which simulates realworld physics with high fidelity, also developed by Open robotics. This software is strongly integrated with ROS and will help in visualising the robot and testing the code before fabricating the real robot, reducing risk of damaging equipment.

RVIZ2 is a 3D visualization tool for ROS2. It allows the user to view the simulated robot model, log sensor information from the robot's sensors, and replay the logged sensor information.

Ubuntu 20.04 LTS (Long-Term Support) is a version of the Ubuntu operating system that is supported for a longer period of time (5 years) than a standard release (9 months). It is also a more stable release, and is recommended for use in production environments. Ubuntu is required to develop in ROS2 and Gazebo.

To run Ubuntu there are 3 methods, first is to use a virtual machine through windows, Windows linux subsystem (WSL2), or to install Ubuntu 20.04 desktop. Out of these options a virtual machine running off windows has the worst performance, whilst WSL2 and native Linux provide similar performance.

Windows Subsystem for Linux (WSL) is a feature in Windows 11 that allows users to run Linux command-line tools on Windows. WSL2 is the second version of WSL, which was first introduced in Windows 11, it uses a lightweight virtual machine (VM) to run the Linux kernel, which provides improved performance and compatibility compared to WSL1. Additionally, WSL2 supports running Linux GUI applications and access to USB devices, and it allows users to run Linux and Windows applications side-by-side.

3.3.2.1 Installing Ubuntu through windows

Installing Ubuntu through Windows 11 was done simply by downloading Ubuntu 20.04 from the Microsoft app store, WSL2 comes presinstalled with Windows 11 and can be enabled through powershell. After testing, this method indeed runs ROS, and Linux GUI applications like Gimp, however an error related to video drivers will be constantly recieved when trying to run RVIZ2 and Gazebo, both essential simulators for this study.

3.3.2.1 Dualboot Ubuntu

As WSL2 failed to run the required simulation softwares, Ubuntu desktop was installed, this is done by first partitioning 50GB of space throguh windows disk manager, and creating the flash USB containing Ubuntu 20.04 desktop. After setup is complete, run commands; ‘sudo apt update’ and ‘sudo apt upgrade’ This will update Ubuntu to the latest packages.

3.3.2.3 Installing ROS2 and Gazebo

Installing ROS was done by running the following code with same order in terminal:

```
sudo apt install software-properties-common


sudo add-apt-repository universe

sudo apt update && sudo apt install curl

sudo curl -sSL
https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o
/usr/share/keyrings/ros-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/ros-archive-keyring.gpg]
http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo
$UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list
> /dev/null

sudo apt update

sudo apt upgrade

sudo apt install ros-foxy-desktop python3-argcomplete
```

Since ROS2 works by communicating between nodes, two instances of the program running on the same network will produce errors, this problem was resolved by connecting raspberry pi and development PC to personal hotspot rather than public WiFi network.

With each new terminal node on Linux, ROS2 has to be defined before it can be used using:
source /opt/ros/foxy/setup.bash
Gazebo is much simpler to install only requiring one line of code:
‘curl -sSL http://get.gazebosim.org | sh’

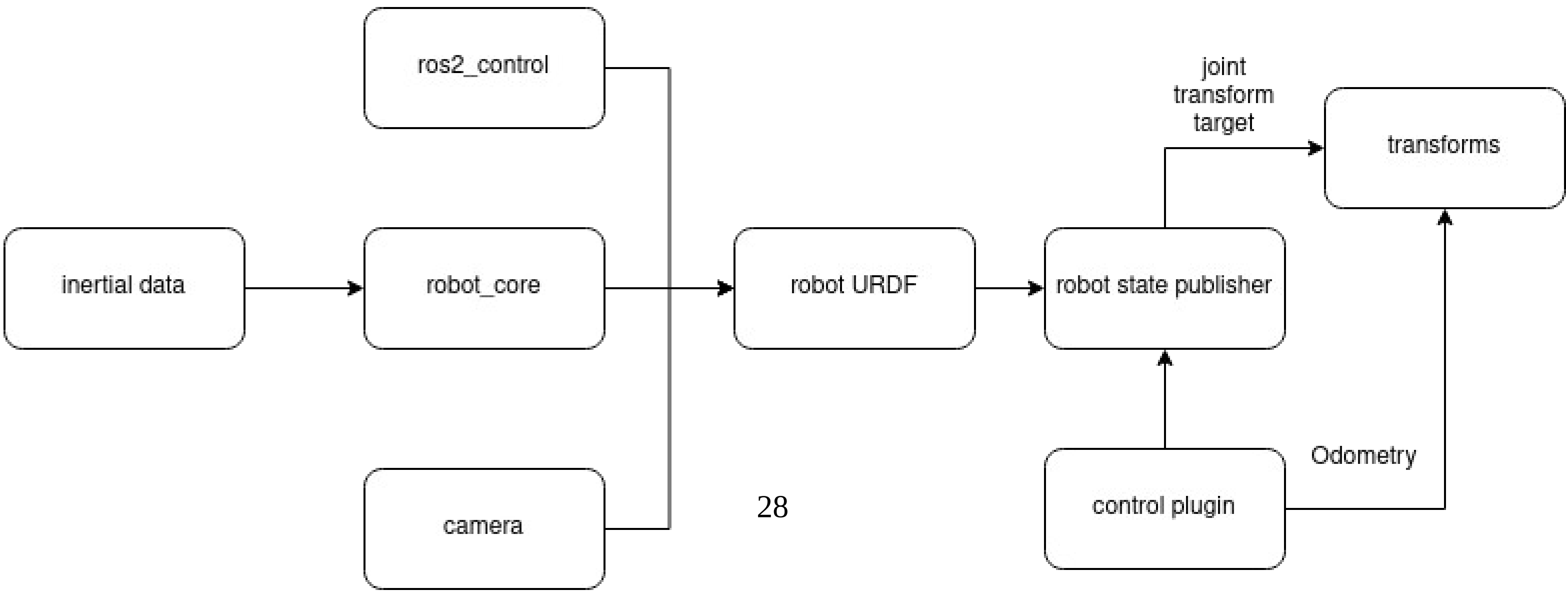
3.3.2.4 Setting up raspberry pi

To setup Raspberry Pi as robot controller it needs to run ROS using ubuntu 20.04 server version, this is important as desktop edition requires more resources. This was done by using Pi imager to flash Ubuntu, then connecting the Raspberry Pi to mobile hotspot. When the development PC and Pi are both connected to same network, simply run the code ‘ssh pi@(IP address of the raspi)’, this allows control of pi through the PC. The IP address of Raspberry Pi can be found using network scanner applications. After this step, simply repeat the steps of downloading ROS2, without Gazebo. The simulator is not needed in the real robot and it will take significant resources to run.

3.3.3 ROS software development

The ROS2 program consitst of a main URDF file which describes the robot’s variables and its controllers, this then communicate with diffrential drive robot control plugins and hardware interface to output target joint transform velocities to the arduino. Figure 3.7 summarises the process in a chart format.

Figure 3.7: Flowchart of the program and its interaction between scripts.



3.3.3.1 Creating ros2 package and linkning to github repository

In order to create a ROS2 package a ~/dev_ws/src/ is first created, followed by the command ‘ros2 pkg (name)’. This package contains a launch, config, and description file. A github repository of is then created based off this package. The reason for doing this is that connecting to github via SSH allows for seamless code update between the development PC and Raspberry Pi.

3.3.3.2 Creating URDF code for mobile robot

URDF (Unified Robot Description Format) is an XML file format used in ROS2 to represent the kinematic and dynamic properties of a robot. URDF files describe the robot's links, joints, and sensors, and are used by ROS to simulate and control the robot. This file is required to create a model and collision behavior for simulation in RVIZ and Gazebo. The URDF is saved as an Xacro file, this will then pe processed by robot_state_publisher, which then broadcast the target joint transforms for the robot to perform.

The code used to describe the mobile robot is as follows:

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


<robot xmlns:xacro="http://www.ros.org/wiki/xacro" >


<xacro:include filename="inertial_macros.xacro"/>
<material name="white">
<color rgba="1 1 1 1" />
</material>

<material name="orange">
<color rgba="1 0.3 0.1 1"/>
</material>

<material name="blue">
<color rgba="0.2 0.2 1 1"/>
</material>

<material name="black">
```

```
<color rgba="0 0 0 1"/>
</material>

<!-- BASE LINK -->
<link name="base_link">
</link>
<!-- CHASSIS LINK -->
<joint name="chassis_joint" type="fixed">
<parent link="base_link"/>
<child link="chassis"/>
<origin xyz="-0.1 0 0"/>
</joint>
<link name="chassis">
<visual>
<origin xyz="0.15 0 0.075"/>
<geometry>
<box size="0.3 0.3 0.15"/>
</geometry>
<material name="white"/>
</visual>
<collision>
<origin xyz="0.15 0 0.075"/>
<geometry>
<box size="0.3 0.3 0.15"/>
</geometry>
</collision>
<xacro:inertial_box mass="0.5" x="0.3" y="0.3" z="0.15">
<origin xyz="0.15 0 0.075" rpy="0 0 0"/>
</xacro:inertial_box>
</link>
<gazebo reference="chassis">
<material>Gazebo/White</material>
</gazebo>
<!-- LEFT WHEEL LINK -->

<!-- RIGHT WHEEL LINK -->
<joint name="right_wheel_joint" type="continuous">
<parent link="base_link"/>
<child link="right_wheel"/>
<origin xyz="0 -0.175 0" rpy="${pi/2} 0 0" />
<axis xyz="0 0 -1"/>
</joint>
<link name="right_wheel">
<visual>
<geometry>
<cylinder radius="0.05" length="0.04"/>
</geometry>
<material name="blue"/>
</visual>
<collision>
<geometry>
```

Refer type, connections & relative position

Refer type, connections & relative position

Collision rigidbody

Pull inertial data

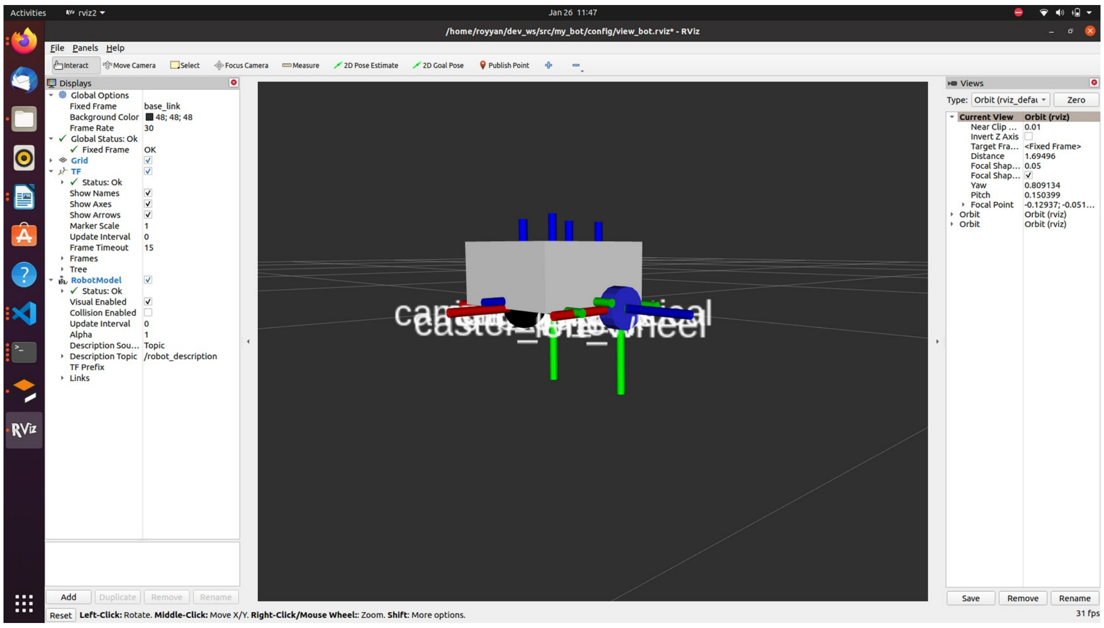
```
<cylinder radius="0.05" length="0.04"/>
</geometry>
</collision>
<xacro:inertial_cylinder mass="0.1" length="0.04" radius="0.05">
<origin xyz="0 0 0" rpy="0 0 0"/>
</xacro:inertial_cylinder>
</link>
<gazebo reference = "right_wheel">
<material>Gazebo/Blue</material>
</gazebo>
<!-- CASTER WHEEL LINK -->
<joint name="caster_wheel_joint" type="fixed">
<parent link="chassis"/>
<child link="caster_wheel"/>
<origin xyz="0.24 0 0"/>
</joint>
<link name="caster_wheel">
<visual>
<geometry>
<sphere radius="0.05"/>
</geometry>
<material name="black"/>
</visual>
<collision>
<geometry>
<sphere radius="0.05"/>
</geometry>
</collision>
<xacro:inertial_sphere mass="0.1" radius="0.05">
<origin xyz="0 0 0" rpy="0 0 0"/>
</xacro:inertial_sphere>
</link>
<gazebo reference = "caster_wheel">
<material>Gazebo/Black</material>
<mu1 value = "0.001"/>
<mu2 value = "0.001"/>
</gazebo>
</robot>
```

To explain this code, first, the colors were specified, followed by each component of the robot which are; base link, chasis, weheel joints, and caster. Base link simply tells RVIZ how the wheel joints are expected to be connected. Inertia calculations were also performed to ensure the simulation response is more similar to the real robot, below is the inertia code for the sphere used in the above code.

```
<xacro:macro name="inertial_sphere" params="mass radius *origin">
<inertial>
<xacro:insert_block name="origin"/>
```

```
<mass value="${mass}" />
<inertia ixx="${(2/5) * mass * (radius*radius)}" ixy="0.0" ixz="0.0"
iyy="${(2/5) * mass * (radius*radius)}" iyz="0.0"
izz="${(2/5) * mass * (radius*radius)}" />
</inertial>
</xacro:macro>
```

Figure 3.8 shows the visualisation of the produced robot in rviz.



```
Global Options
  Fixed Frame           base_link
  Background Color      48; 48; 48
  Frame Rate            30
Global Status: Ok
  Fixed Frame           OK
Grid                    ✓
TF                      ✓
  Status: Ok
  Show Names            ✓
  Show Axes             ✓
  Show Arrows           ✓
  Marker Scale          1
  Update Interval       0
  Frame Timeout         15
  Frames
  Tree
RobotModel              ✓
  Status: Ok
  Visual Enabled        ✓
  Collision Enabled     ☐
  Update Interval       0
  Alpha                 1
  Description Sou...    Topic
  Description Topic     /robot_description
  TF Prefix
  Links
```

```
Type: Orbit (rviz_defau
Current View            Orbit (rviz)
  Near Clip ...         0.01
  Invert Z Axis         ☐
  Target Fra...         <Fixed Frame>
  Distance              1.69496
  Focal Shap...         0.05
  Focal Shap...         ✓
  Yaw                   0.809134
  Pitch                 0.150399
  Focal Point           -0.12937; -0.051...
Orbit                   Orbit (rviz)
Orbit                   Orbit (rviz)
```

```
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
```

robot_core script essentially describes the robot’s shape, mass, interia, for simulation. The visual representaion of the code can be checked in RVIZ2 to verify the simulated design of the robot. Figure 3.8 shows the realy results from RVIZ2, note no simulated camera was added yet.

3.3.3.3 Control system of robot using ros2_control

The control system of the robot was developed using ros2_control library, installed using the code ‘sudo apt install ros-foxy-ros2-control ros-foxy-ros2-controllers ros-foxy-gazebo-ros2-control’ in command line. Ros2_control allows fast interfacing between the controllers and hardware interface. To use this library, a differential drive controller and joint state broadcaster package was specified, with required parameters set based on the URDF simulated robot.

```
controller_manager:


ros__parameters:


update_rate: 30
use_sim_time: true

diff_cont:
type: diff_drive_controller/DiffDriveController
joint_broad:
type: joint_state_broadcaster/JointStateBroadcaster

diff_cont:
ros__parameters:
publish_rate: 50.0
base_frame_id: base_link
left_wheel_names: ['left_wheel_joint']
right_wheel_names: ['right_wheel_joint']
wheel_separation: 0.35
wheel_radius: 0.05
use_stamped_vel: false
```

Specifies the packages to use

Inputting the required parameters for robot

A ros2 control Xacro file is also created to specify the velocity limits for the robot to perform. The code below does this function.

```
<ros2_control name="GazeboSystem" type="system">


<hardware>
<plugin>gazebo_ros2_control/GazeboSystem</plugin>
</hardware>
<joint name="left_wheel_joint">
<command_interface name="velocity">
<param name="min">-10</param>
<param name="max">10</param>
</command_interface>
<state_interface name="velocity"/>
<state_interface name="position"/>
</joint>
<joint name="right_wheel_joint">
<command_interface name="velocity">
<param name="min">-10</param>
<param name="max">10</param>
</command_interface>
<state_interface name="velocity"/>
```

```
<state_interface name="position"/>
</joint>
</ros2_control>
```

Lastly, for ros2 control to work with gazebo and real robots, the previously specified packages DiffDriveController and JointStateBroadcaster need to be installed on the computer, then run using ros2 run controller_manager spawner.py (package name), before launching gazebo.

3.3.3.4 Simulating in Gazebo

To simulate the robot in Gazebo the URDF has to be modified with <gazebo> tags, thiss is because the colours in URDF do not translate in Gazaebo, and the reult will simply be a black robot with the correct shape.

```
<gazebo reference="chassis">
<material>Gazebo/White</material>
</gazebo>
```

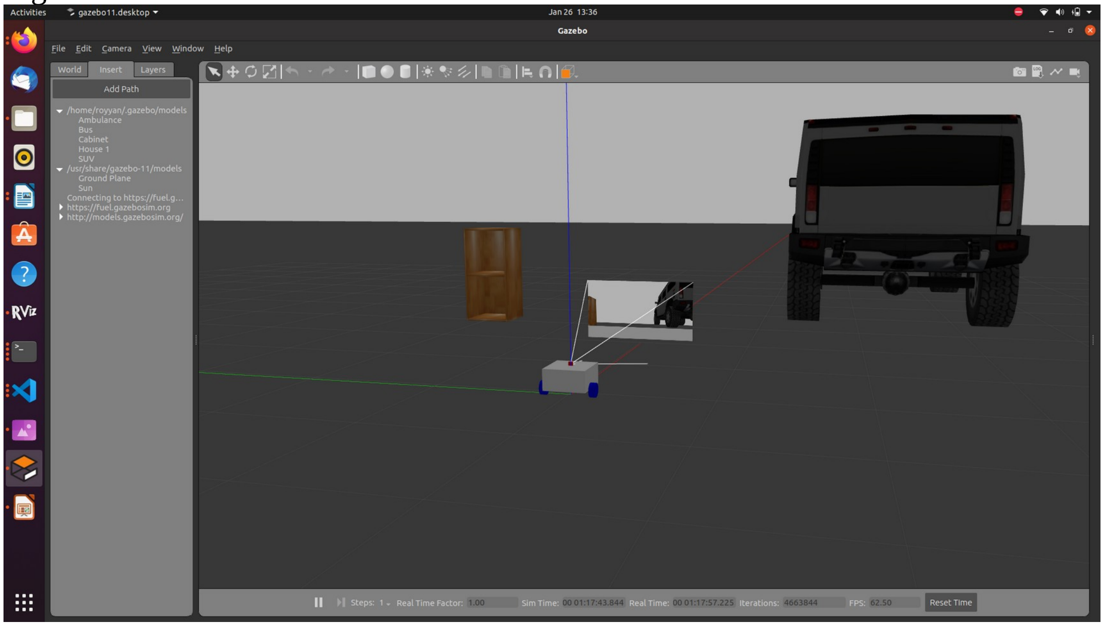
The above code was added for every component specified in the URDF file, afterwhich a launch file was made using python, this is because the code for launching a specific model and rebuilding in gazebo through terminal is too burdensome. The code used is as follows;

```
package_name='my_bot'
rsp = IncludeLaunchDescription(
PythonLaunchDescriptionSource([os.path.join(
get_package_share_directory(package_name),'launch','rsp.launch.py'
)]), launch_arguments={'use_sim_time': 'true'}.items()

# Include the Gazebo launch file, provided by the gazebo_ros package
gazebo = IncludeLaunchDescription(
PythonLaunchDescriptionSource([os.path.join(
get_package_share_directory('gazebo_ros'), 'launch', 'gazebo.launch.py')]),

# Run the spawner node from the gazebo_ros package.
spawn_entity = Node(package='gazebo_ros', executable='spawn_entity.py',
arguments=['-topic', 'robot_description',
'-entity', 'my_bot'],
output='screen')
```

Now with an appropriate launch file, launching the simulation from a fresh terminal node only requires a few lines;

“source /opt/ros/foxy/setup.bash **<-refers ROS2 path**
cd dev_ws **<- refers the project file directory**
source install/setup.bash
colcon build –symlink-install **<-rebuild project incase changes were made to code**
ros2 launch my_bot launch_sim.launch.py” **<-launch ROS node and gazebo**

Figure 3.9: Simulation of Robot in Gazebo with simulated camera feedback.


Gazebo
File Edit Camera View Window Help
World Insert Layers
Add Path
/home/royyan/.gazebo/models
Ambulance
Bus
Cabinet
House 1
SUV
/usr/share/gazebo-11/models
Ground Plane
Sun
https://fuel.gazebosim.org
http://models.gazebosim.org/
Reset Time

3.3.3.5 Simulating Camera module

Camera module is added by creating a new Xacro URDF file, it works just as a chasis was described, a bot with the inertia and collision description included, but this time add a gazebo plugin to declare this a camera sensor.

```
<plugin name="camera_controller" filename="libgazebo_ros_camera.so">
<frame_name>camera_link_optical</frame_name>
</plugin>
```

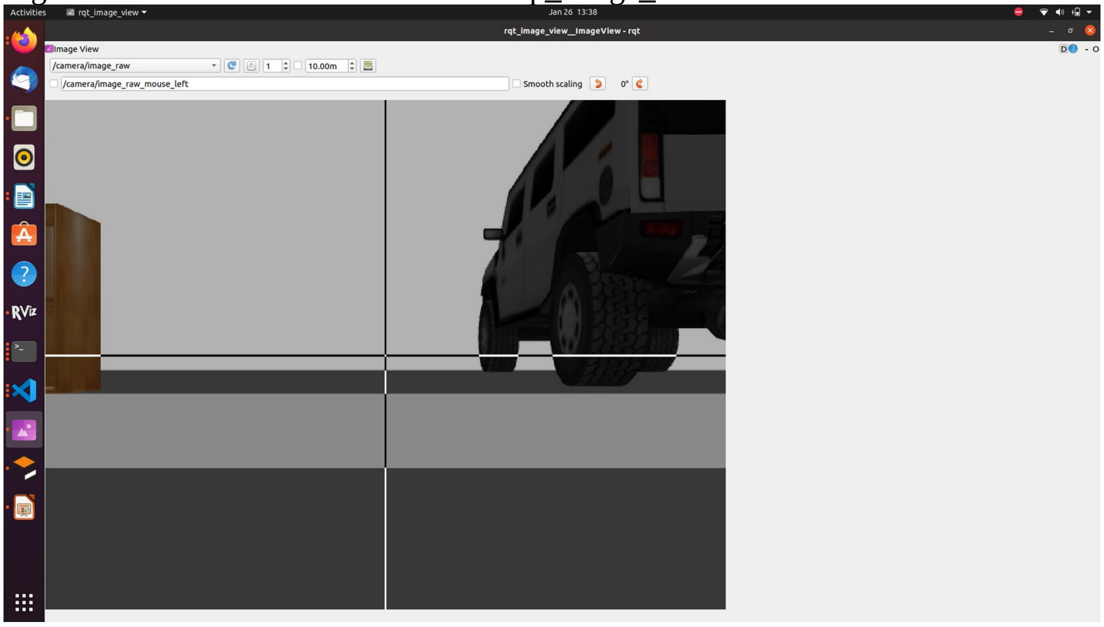
The camera’s specification and range has to also be specified in this URDF file.

```
<camera>
<horizontal_fov>1.089</horizontal_fov>
<image>
<format>R8G8B8</format>
<width>640</width>
<height>480</height>
</image>
<clip>
<near>0.05</near>
<far>8.0</far>
</clip>
</camera>
```

Camera resolution specified here

After launching Gazebo, the images from the camera can be seen by opening another ROS node. Figure 3.10 shows the camera feed the robot outputs through rqt_image_view, note that any computer in the same network running a ros node can recieve this data. It is important for rqt_image_view to output simulated camera data before building the robot because this same method will be used to automatically pull camera image data from Raspberry Pi to the development computer

Figure 3.10: Simulated Camera feel on rqt_image_view



3.4 Hardware Development

This section covers the hardware development aspect to produce the working prototype, it involves planning the components, configuring the parts, and lastly testing the whole electronic system before assembling the robot. Figure 3.11 shows the flowchart of hardware development methodology.

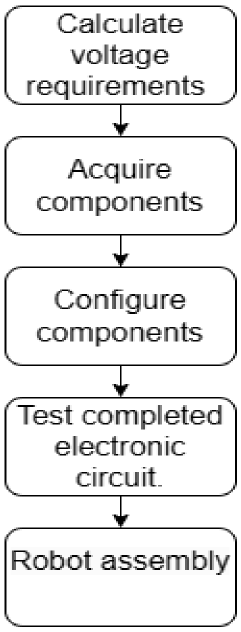
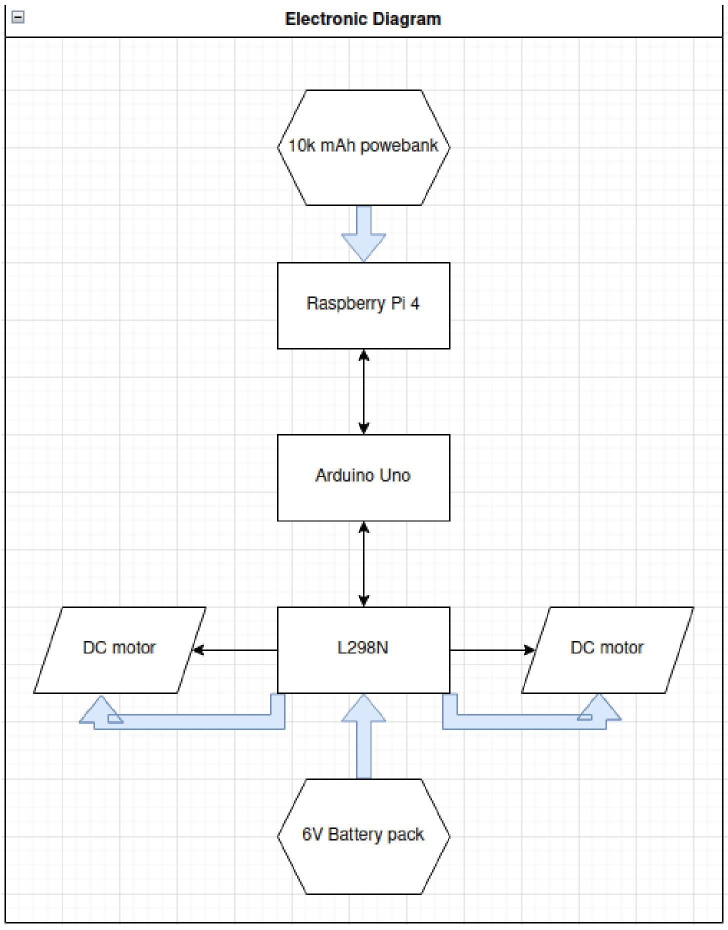

Calculate voltage requirements
Acquire components
Configure components
Test completed electronic circuit.
Robot assembly

Figure 3.11: Flowchart of hardware development methodology.

3.4.1 System Diagram

Figure 3.12 shows the electronic system implemented, the black arrows represent data transfer, whilst the blue arrows represemt power transfer. The robot uses two power sources, powewr bank for raspberry pi and arduino, and a 6V battery pack to drive the motors. The Arduino is powered through the USB port of the Raspberry Pi. In this setup the Pi works as the robot controller and camera node for ROS, the Arduino recieves target velocities for the two wheels, and outputs it to the L298N motor driver. Figure 3.13 shows the electronic system fabricated.
The electronics of the mobile robot are tested before connecting to the chaisis, the camera functions normally and the motors turn similarly to the motor positoons displayed in RVIZ2, this is controlled via keyboard from development PC.
Figure 3.14 shows the robot chasis, which is an affortable car kit.

Figure 3.12: Electronic sysstem implemented for robot.



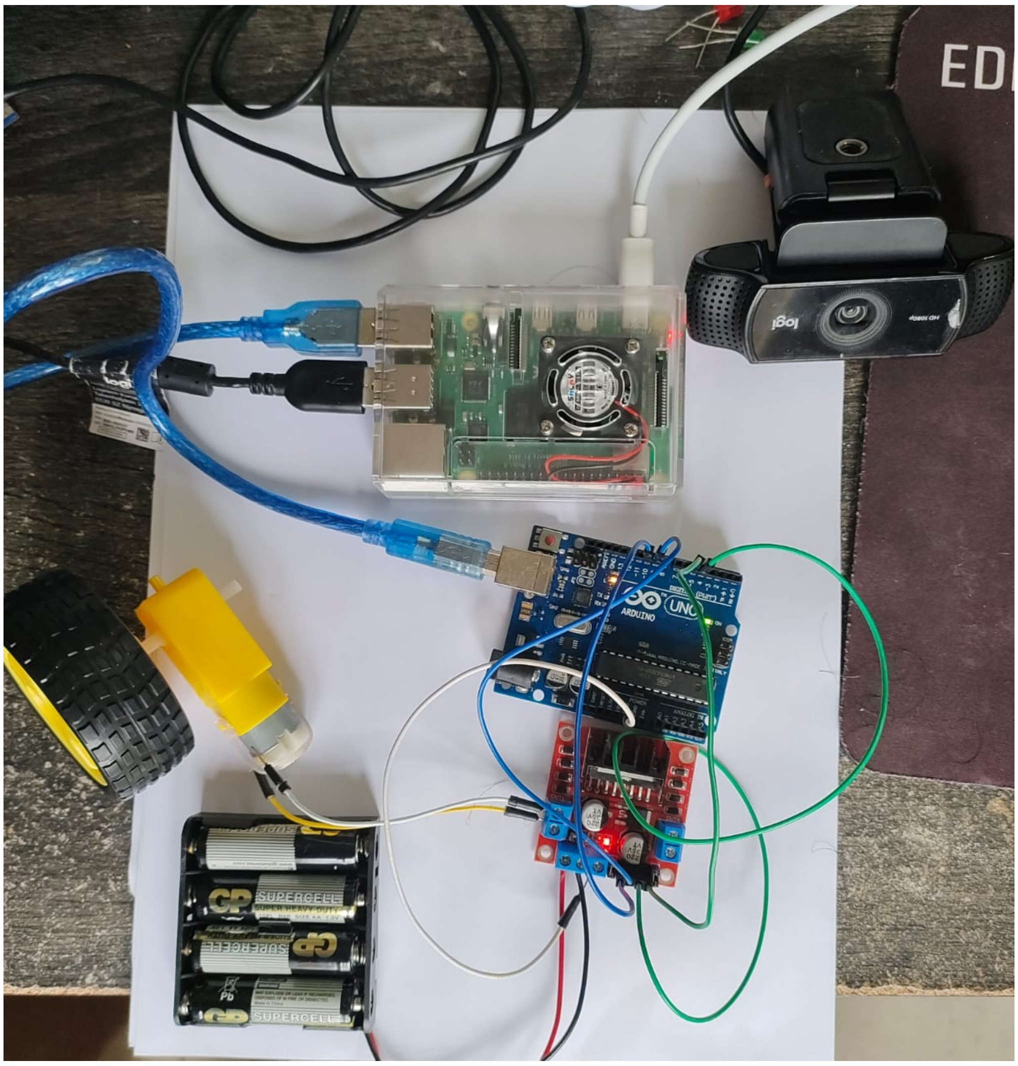
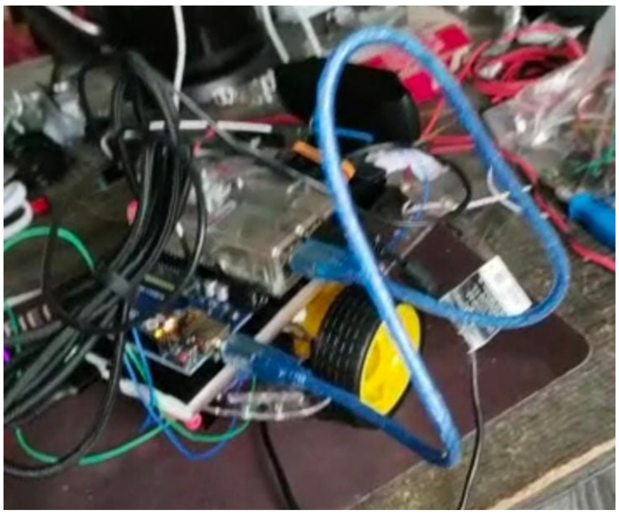
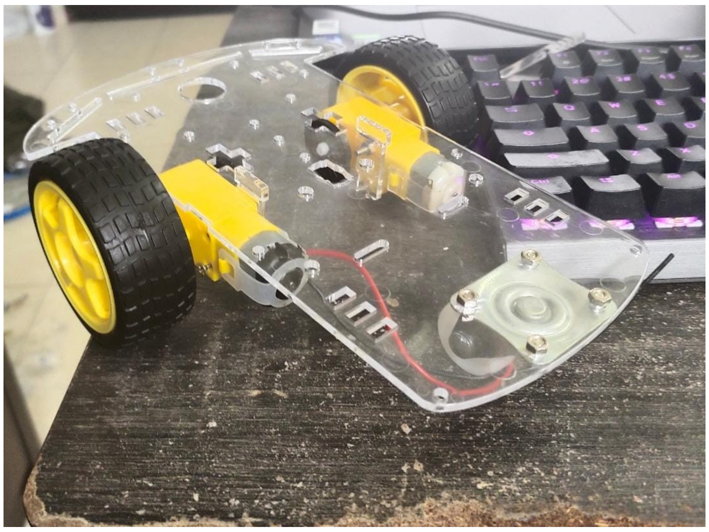


Figure 3.13: Electronics components fabricated.

Figure 3.14: Chasis of mobile robot.

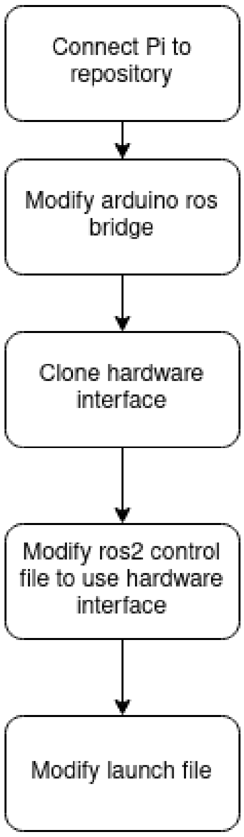




3.5 Hardware-Software Integration

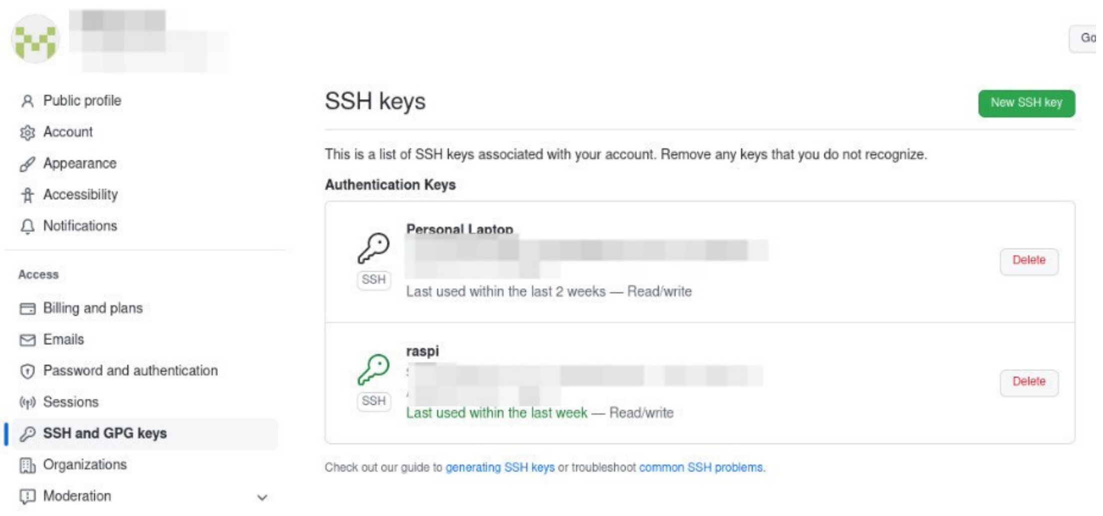
This section covers the process of integrating the working hardware with software to produce a functional prototype UGV. Figure 3.15 shows a flowchart of Hardware-Software integration methodology. It involves setting up the Raspberry. Arduino. and modifying the code to work with hardware interface

Figure 3.15: Flowchart of Hardware-Software integration methodology.


Connect Pi to repository
Modify arduino ros bridge
Clone hardware interface
Modify ros2 control file to use hardware interface
Modify launch file

3.5.1 Connecting to raspberry pi

The development computer needs to connect to raspberry pi to install ros package and pull the project from git. An IP scanner was used to find the address of the Pi, and the command ssh pi@(IP address) was used on the dev machine to connect to the raspberry. To clone the project files, an SSH connection to github has to first be established to github, this is done through personal setting in github. After which the project files can be downloaded using the code; ‘git clone (link to repository)‘.



SSH keys

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

Authentication Keys

Personal Laptop
[redacted]
Last used within the last 2 weeks — Read/write

raspi
[redacted]
Last used within the last week — Read/write

Check out our guide to generating SSH keys or troubleshoot common SSH problems.

It is impotant to reduce the workload of the pi in order to maximize the performance of the camerra, so the code used in the raspberry was modified to remove all gazebo tags, as simulation is uneccessary.

3.5.2 Connecting to real camera through raspberry pi

Once the Pi is connected through SSH it only needs a few apps to feed camera to webcam.

sudo apt install ros-foxy-rqt_image_view

This will install the image viewer application.

sudo apt install libraspberrypi-bin v4l-utils ros-foxy-v4l2-camera ros-foxy-image-transport-plugins

This command installs libraries to use v4l2, of video for linux.

After plugging in the camera, its port identification has to first be noted, it can be found using ‘v4l2-ctl –list-devices’, this is important as ROS will not know which camera to launch.

The ros command to launch the camera is;

ros2 run v4l2_camera v4l2_camera_node --ros-args -p video_device:="/dev/video2" -p image_size:="[1920,1080]" -p camera_frame_id:=camera_optical_link

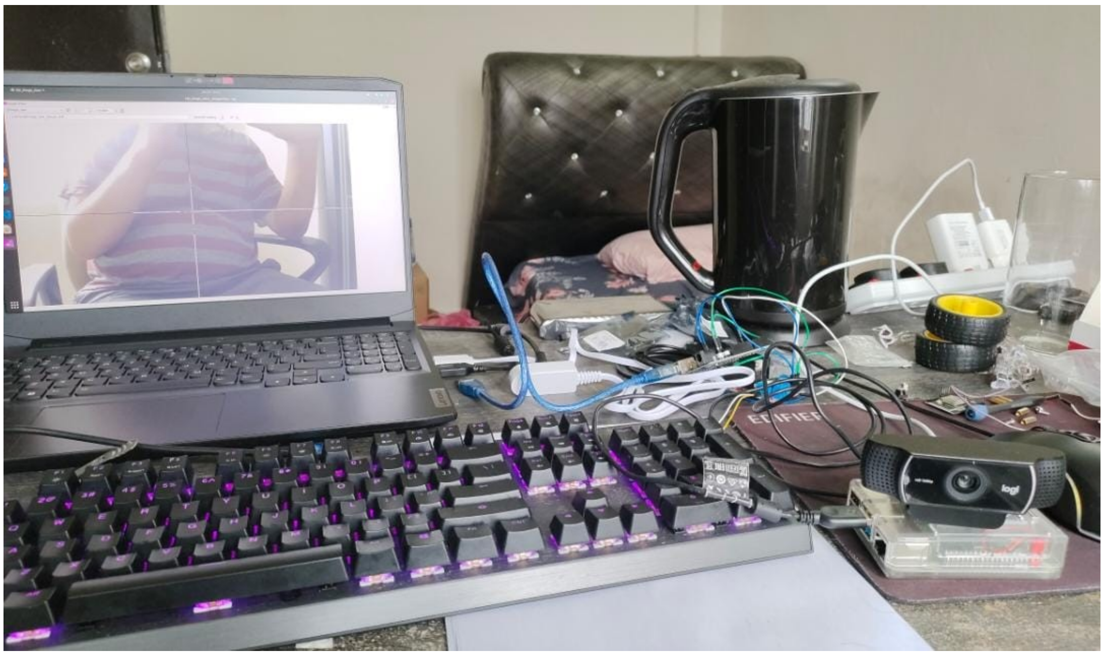
Understanding how to modify this command is important as it allows to tune camera settings based on computer limitations, in this case 1280*720 was the limit.

More details on this command as follows;

- video_device:=”” referances the webcam to use, by default it is /dev/video0
- image_size:=”[]” specifies the camera resolution.

Figure 3.16 shows the raspberry pi connected to webcam feeding image data through network on development computer.

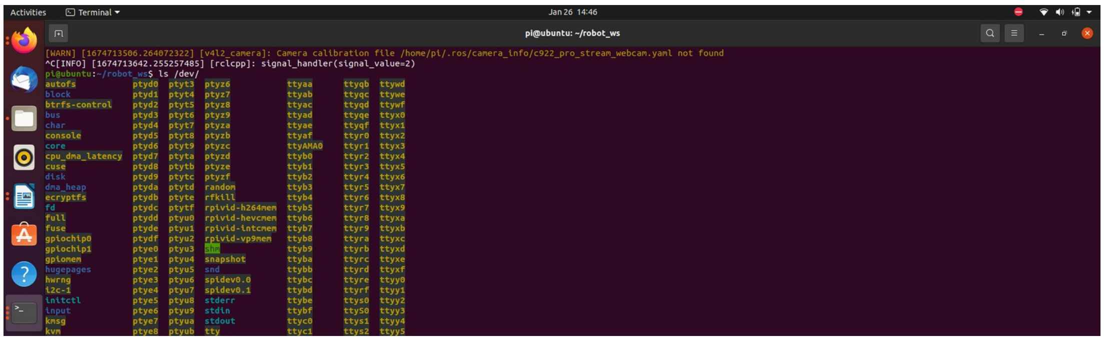
Figure 3.16 :Raspberry Pi feeding camera image to development PC



3.5.3 Integrating arduino as a motor controller

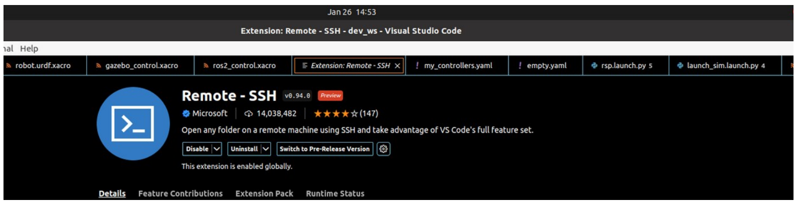
To interface Arduino with ROS2, an existing arduino sketch called ros_arduino_bridge developed by hbrobotics is used, this can be downloaded from ROS official website. The target pins must be modified on arduino IDE to suit the circuit as per in figure 3.13, only then upload the sketch to the Arduino.

Once the Raspberry Pi is connected to arduino via serial it is important to find the identification of the arduino in linux, to do this *ls /dev/* waas input to find the serial port coming from arduino, in this study it was /dev/ACM0. This information allows us to check if the circuit is working. By inputting miniterm -e /dev/ACM0 57600 in terminal, the DC motors can be controller by manually inputting velocity into the command line.



```
[WARN] [1674713506.264072322] [v4l2_camera]: Camera calibration file /home/pi/.ros/camera_info/c922_pro_stream_webcam.yaml not found
^C[INFO] [1674713642.255257485] [rclcpp]: signal_handler(signal_value=2)
pi@ubuntu:~/robot_ws$ ls /dev/
autofs           ptyd0  ptyt3  ptyz6           ttyaa    ttyqb  ttywd
block            ptyd1  ptyt4  ptyz7           ttyab    ttyqc  ttywe
btrfs-control    ptyd2  ptyt5  ptyz8           ttyac    ttyqd  ttywf
bus              ptyd3  ptyt6  ptyz9           ttyad    ttyqe  ttyx0
char             ptyd4  ptyt7  ptyza           ttyae    ttyqf  ttyx1
console          ptyd5  ptyt8  ptyzb           ttyaf    ttyr0  ttyx2
core             ptyd6  ptyt9  ptyzc           ttyAMA0  ttyr1  ttyx3
cpu_dma_latency  ptyd7  ptyta  ptyzd           ttyb0    ttyr2  ttyx4
cuse             ptyd8  ptytb  ptyze           ttyb1    ttyr3  ttyx5
disk             ptyd9  ptytc  ptyzf           ttyb2    ttyr4  ttyx6
dma_heap         ptyda  ptytd  random          ttyb3    ttyr5  ttyx7
ecryptfs         ptydb  ptyte  rfkill          ttyb4    ttyr6  ttyx8
fd               ptydc  ptytf  rpivid-h264mem  ttyb5    ttyr7  ttyx9
full             ptydd  ptyu0  rpivid-hevcmem  ttyb6    ttyr8  ttyxa
fuse             ptyde  ptyu1  rpivid-intcmem  ttyb7    ttyr9  ttyxb
gpiochip0        ptydf  ptyu2  rpivid-vp9mem   ttyb8    ttyra  ttyxc
gpiochip1        ptye0  ptyu3  shm             ttyb9    ttyrb  ttyxd
gpiomem          ptye1  ptyu4  snapshot        ttyba    ttyrc  ttyxe
hugepages        ptye2  ptyu5  snd             ttybb    ttyrd  ttyxf
hwrng            ptye3  ptyu6  spidev0.0       ttybc    ttyre  ttyy0
i2c-1            ptye4  ptyu7  spidev0.1       ttybd    ttyrf  ttyy1
initctl          ptye5  ptyu8  stderr          ttybe    ttys0  ttyy2
input            ptye6  ptyu9  stdin           ttybf    ttyS0  ttyy3
kmsg             ptye7  ptyua  stdout          ttyc0    ttys1  ttyy4
kvm              ptye8  ptyub  tty             ttyc1    ttys2  ttyy5
```

The main project then needs to be modified from outputting data to gazebo instead to ros2_control, but the changes should only take place in the raspberry as gazebo data from the development computer is still nacessary to record. Vscode was installed in Pi and edited via the development machine through remote SSH plugin in vscode.



Open any folder on a remote machine using SSH and take advantage of VS Code's full feature set.

This extension is enabled globally.

The code was modified for the controller manager to launch forcibly without gazebo, and robot description from the URDF file is output manually isntead of using gazebo_ros2_control.

3.6 Chapter Summary

In this chapter, the experiment methodology is discussed. First and foremost, the chosen photogrammetry methodology along with its motivation are discussed. Secondly, the software development process is discussed. Thirdly the component list and its schematic diagram is discussed. Lastly software and hardware is integrated to fabricate the mobile robot.

Chapter 4 Result Discussion

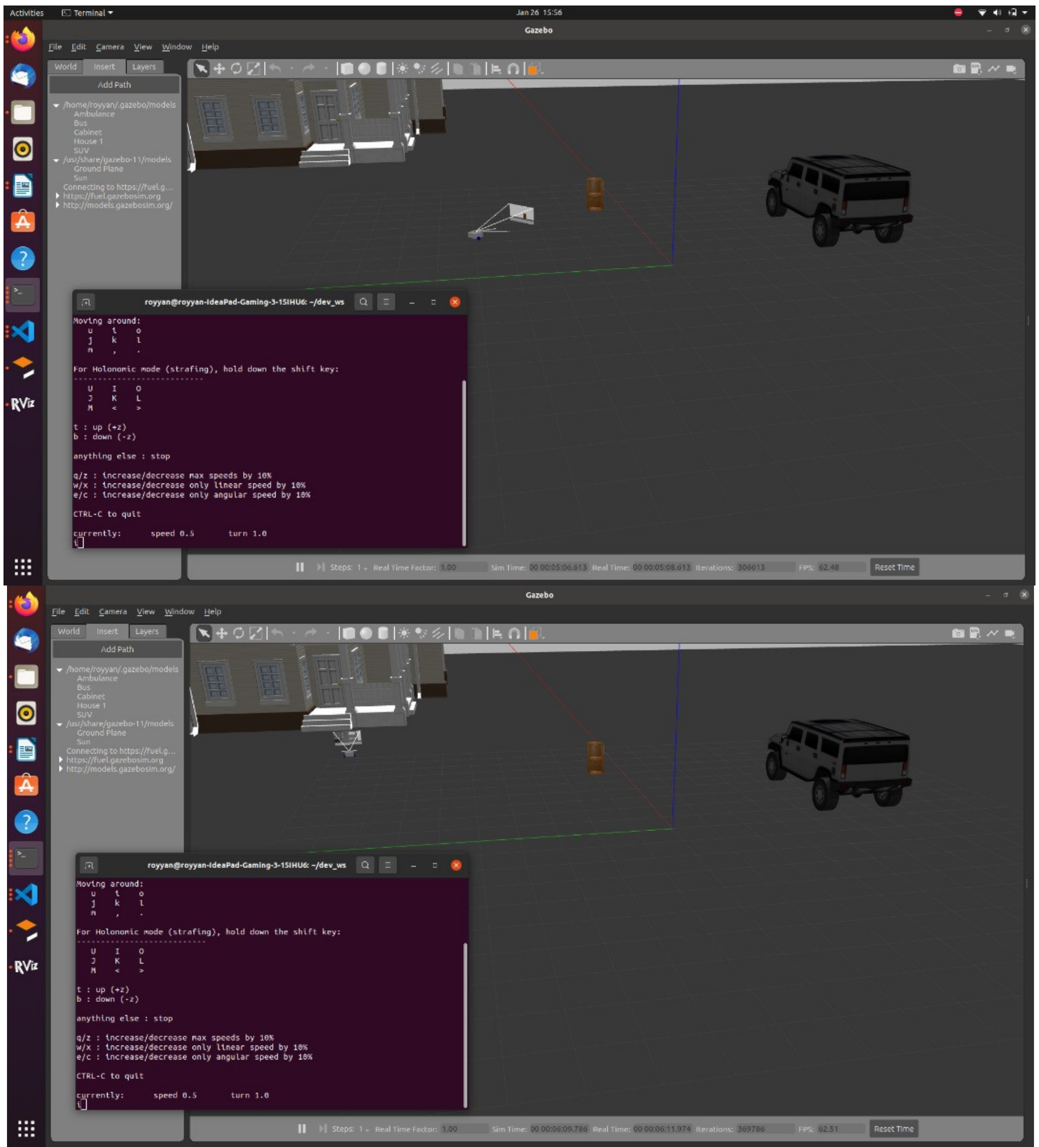
4.1 Preface

The results of this study are presented in this chapter, the robustness of the simulated robot was determined, as well as the camera feed quality recieved from the Raspberry Pi via webcam.

4.2 Driving around simulated robot

The mobile robot was tested via simulation before fabrication, it is controlled via a seperate ROS2 node teleop_twist_keyboard, which allows control from the development PC. The movement is as expected for a differential drive UGV, forward, reverse, and rotational movement works as expected, and its capable of moving around in a circle by rotating and moving forward at same time. The circling radius can be modified by lowering or increasing the angular velocity, increasing this lowers the circling radius until the robot turns on its own axis. Figure 4.1 shows the robot being tested in simulation environment with simulated camera module.

Figure 4.1: Robot Simulated in Gazebo

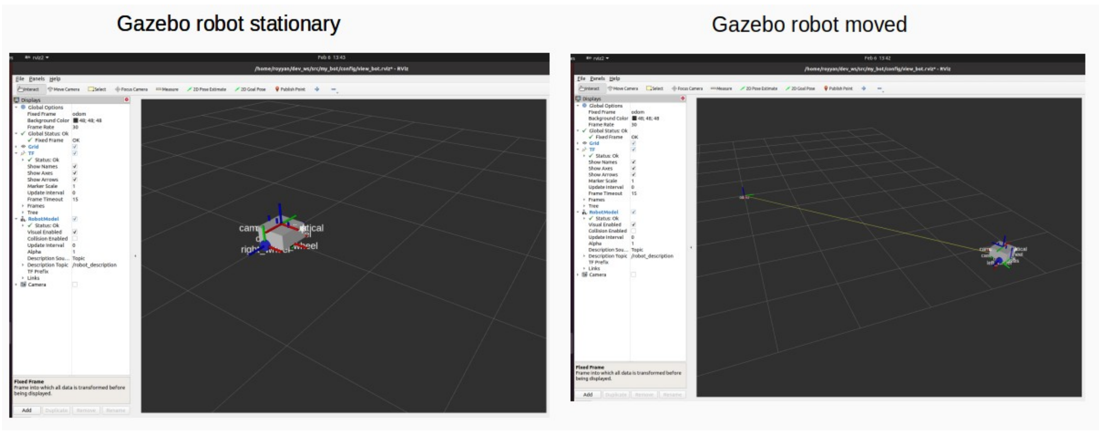

Gazebo
File Edit Camera View Window Help
World Insert Layers
Add Path
royyan@royyan-IdeaPad-Gaming-3-15IHU6: ~/dev_ws
Reset Time

```
Moving around:
   u    i    o
   j    k    l
   m    ,    .

For Holonomic mode (strafing), hold down the shift key:
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently:      speed 0.5       turn 1.0
i
```

```
Moving around:
   u    i    o
   j    k    l
   m    ,    .

For Holonomic mode (strafing), hold down the shift key:
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently:      speed 0.5       turn 1.0
```

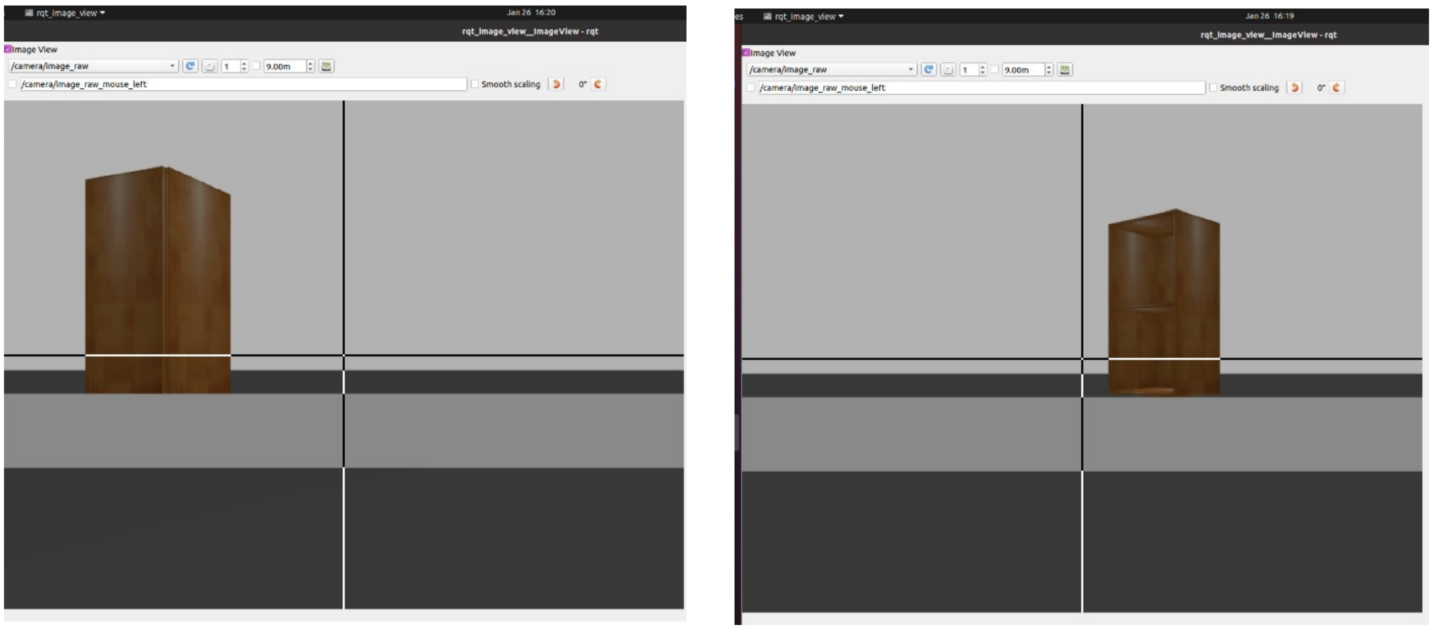
The simulated robot also provided odometry data, this can be used for NAV2 and SLAM autonomous navigation when paired in sensor fusion with 2D LiDAR data. Figure 4.2 shows the robot odometry, as the gazebo simulation moves, the robot in RVIZ changes its displacement and wheel position.

Figure 4.2: RVIZ2 Odometry.



4.3 Testing photos taken by simulation camera

Figure 4.3 Camera feed from rqt_image_view

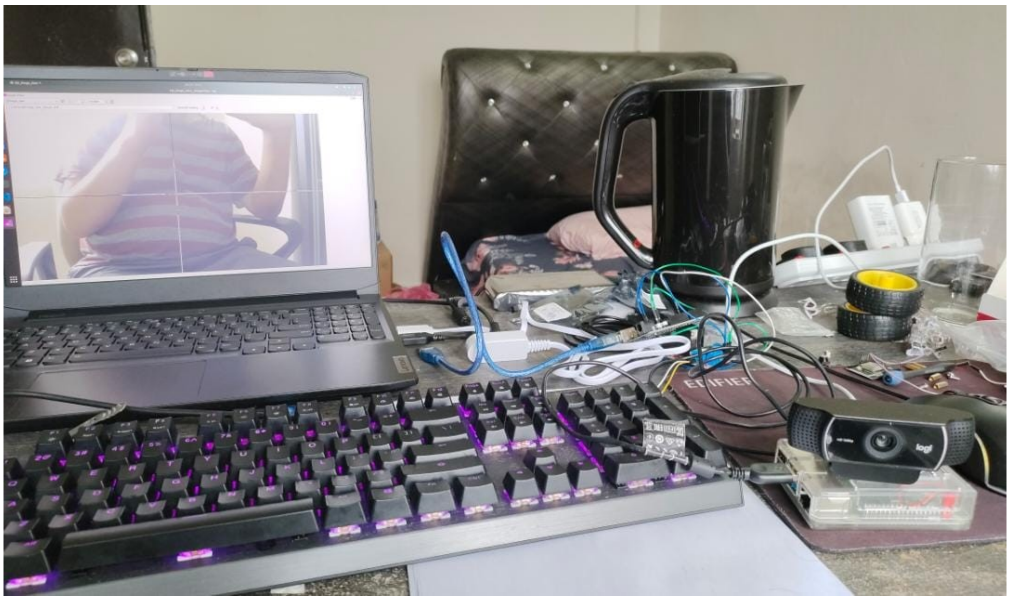


Upon testing the simulation camera, it can be noted that the image raw feed recieeved from gazebo is stable, and accurate to the resolution set in its URDF (in this case 640x480). However one drawback is that manually controlling the robot to take photos can take a lot of time, especially for photogrammetry where a large image dataset it required. This can be offset by automatically timing the camera feed to save the image in fixed time intervals instead of saving it manually, or implementing NAV2, a ROS2 feature for developing autonomous guided vehicles. Figure 4.3 shows the camera data recieved from rqt_image_view, the camera moves as simulation robot is moved.

4.4 Recieving camera data from Pi via network

The Raspberry Pi has successfully output image information to other ROS2 nodes in the same network, from this there are several observations made. When setting the image quality of the webcam at 1920x1080, the Raspberry Pi would output a few frames and crash, so it is unable to output full HD images consistently, however at 1280x720p resolution the raspberry pi outputs 15fps video consistently when image is raw, and 24fps when image is compressed, this means that the quality of images taken by robots are lmited by the microporcessor built in, this case the controller used is a 2GB variant raspberry pi 4b. The main reason for this limitation is that the Raspberry Pi does not come with built in GPU, so image processing is forced to be done on CPU, which is very inefficient. In computer vision dependant robots an Nvidia jetson nano is ofter implemented instead of Raspberry Pi. Figure 4.4 shows the image data from Raspberry being automatically transferred to the Computer.

Figure 4.4: Camera data being recieved on computer from Raspberry Pi



The output images can then be automatically using image_saver plugin from ROS, the timing of camera capture can be artificially produced by controlling the

frequency by which the ros node runs the command. After installing image_view plugin for ROS2, the command for this is;
"ros2 run image_view image_saver image:=image_raw _save_all_image:=false _filename_format:=foo.jpg __name:=image_saver"

It is important to make a directory using mkdir before running this command because all the imaged will be saved in the current directory of the terminal node. Do note this method currently is not supported for ROS2, so performance is not optimized.

4.5 fabricated robot

The robot is fabricated as Figure 4.5 below, all the key electrical components are placed above the robot to prevent damage from vibration, motor driver is placed below with the motor to minimize required wire length used by motor. There is no balancing problems with the robot when configured, however when webcam is removed for troubleshooting, the robot is backheavy due to the 10,000 mAh battery used. The L298N motor driver was placed below the robot because it is a relatively durable component, this component also produces high temperature during operation, so is deliberately seperated from the other components.

Figure 4.5: Fabricated prototype UGV.

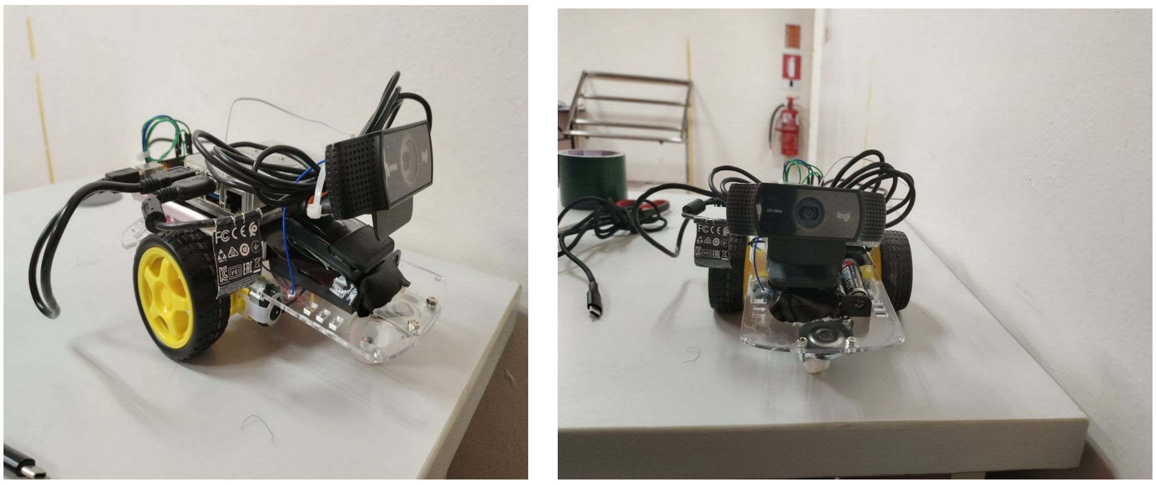
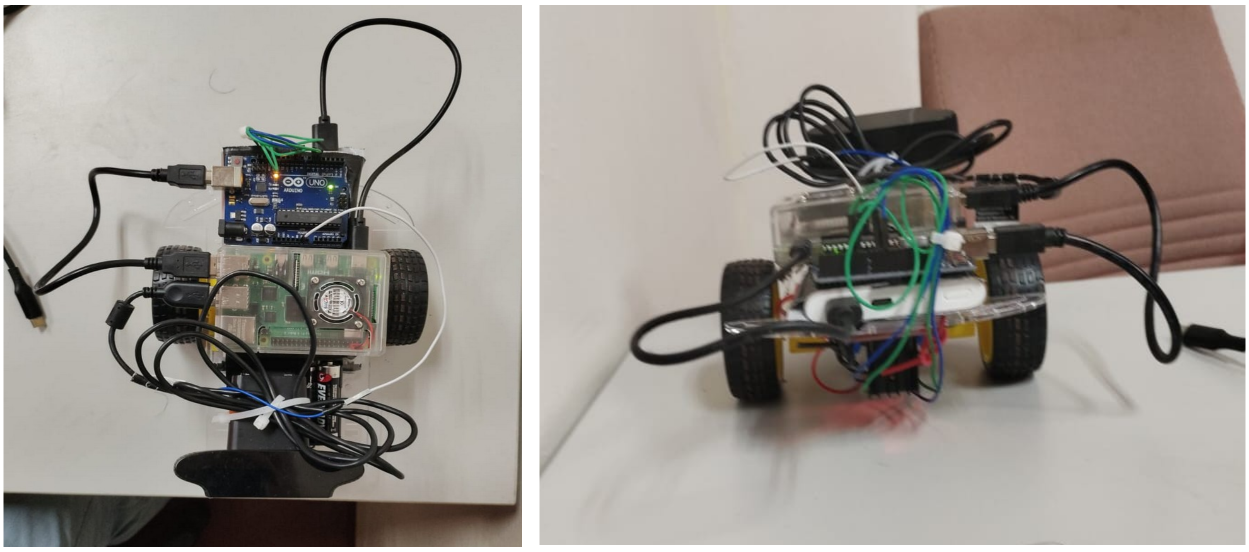


Figure 4.5: Fabricated prototype UGV. Cont.

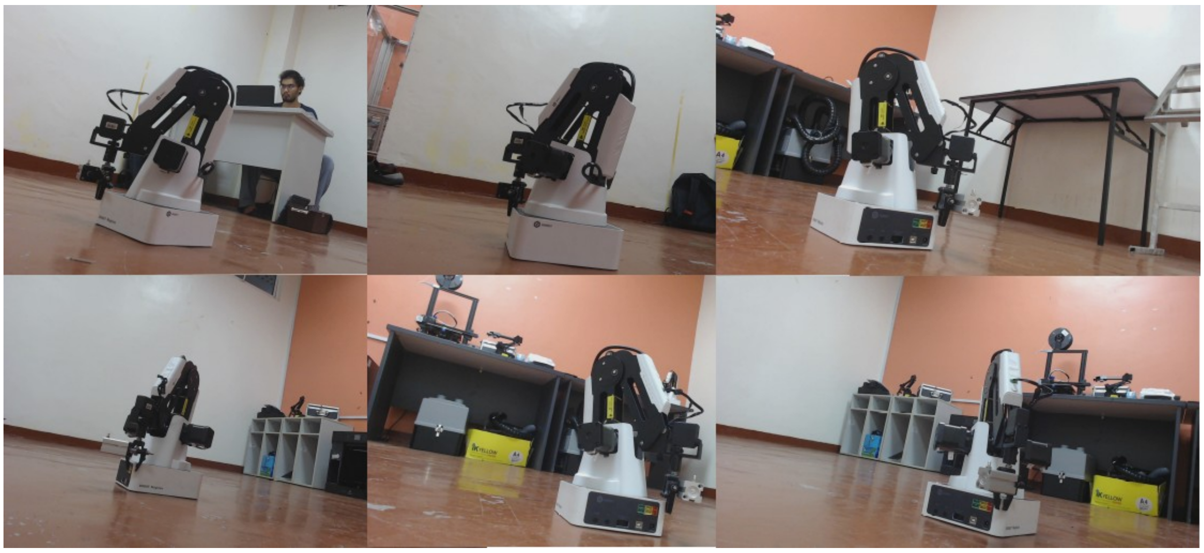


The robot is controlled using a keyboard from the development PC conneced to the raspberry throgh SSH on the same network. Cable management was done by simply tying the wires together with plastic fastener cable.

4.6 Photo results

The robot was tested by experimenting its photogrammetry capabilities, this was conducted in indoor lab environment with controled lighting. The robot is tasked to circle a dobot magician robotic arm and record the image data, the radius of turn is controlled by reducing/increasing angular velocity limits through keyboard controlls, higher angular velocity resultrs in smaller turning radius until the robot turns on its own axis. 45 sample images were taken at a resolution of 1280x720 pixel, but compressed to prevent low framerates. At higher resolutions the images actually end up blurry as the image output is lagging with actual robot movement. Figure 4.6 shows some of the images taken.

Figure 4.6: Image results from UGV.



4.7 Photogrammetry results

The images from the robot were then input into an open-source software called meshroom to produce a 3D mesh. A point cloud of the entire room was first generated by the software, followed by its resulting mesh. The generated mesh was that of the entire room with the robot in the middle, meshlab is then used to remove the room artifacts to just leave the robotic arm. The model was then exported as .obj format and tested on unity game engine with no difficulty. The results in Figure 4.7 shows the dobot magician as a .obj file 3D model, it is significantly better than the results produced from turntable photogrammetry in preliminary study. The results are not as good as those for museum application in literature review [6], however significantly better than the rover mobile robot [3].

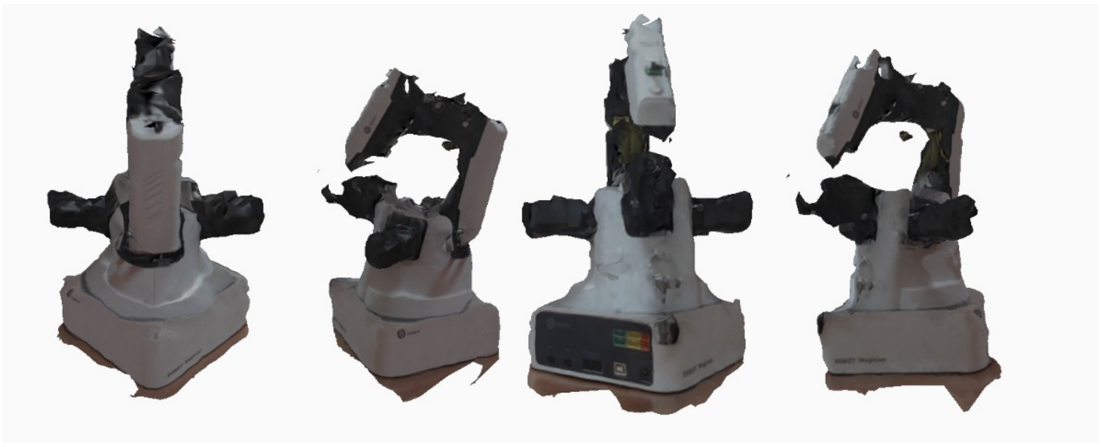


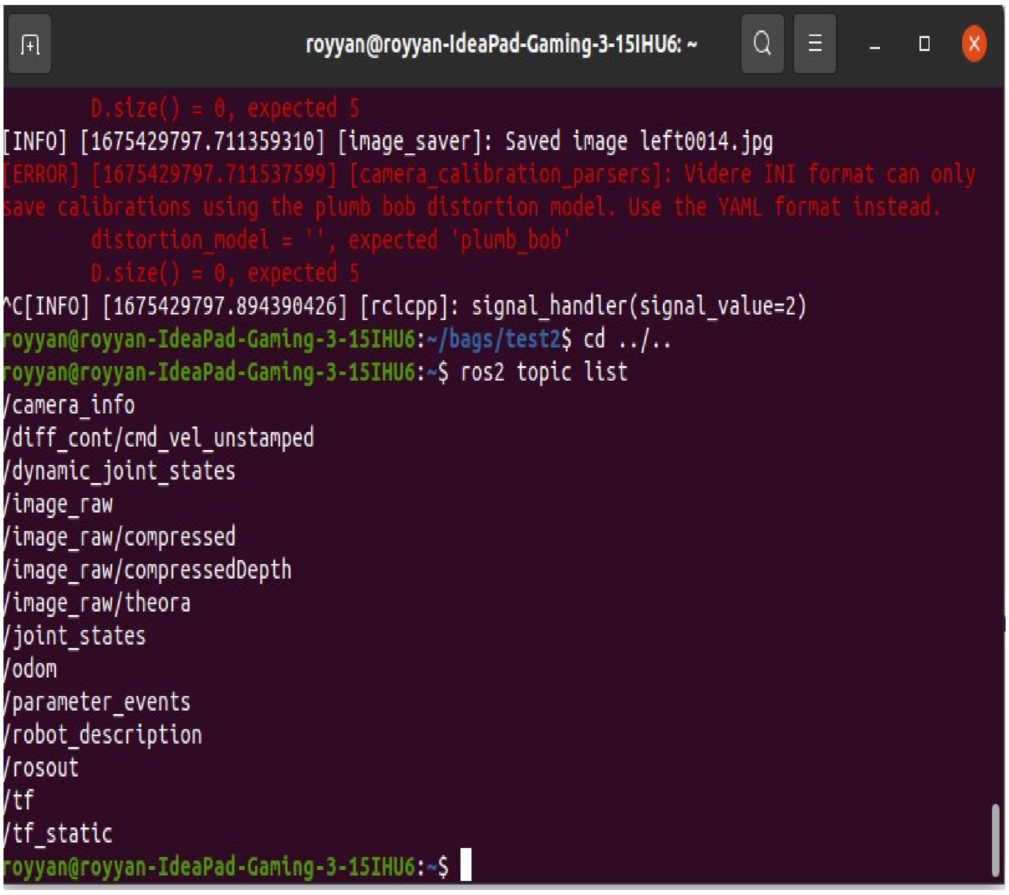
Figure 4.7: Dobot model as result.

The model of Dobot produced has an accurate shape of the Robotic arm, however its end effector gripper was not well reproduced. Textures turned out well, with the dobot logo even reporduced in the model.

4.8 Running ROS nodes

This section covers the ROS2 nodes and topics running when the robot is in operation. Figure 4.8 shows the topic list, these are essentially "published" data which any ROS node on the same network can subscribe to. In this case /image_raw/compressed was used to pull image data, /joint_states provide target joint positions, and /diff_contcmd_vel_unstamped provide target joint velocities to reach target joint position.

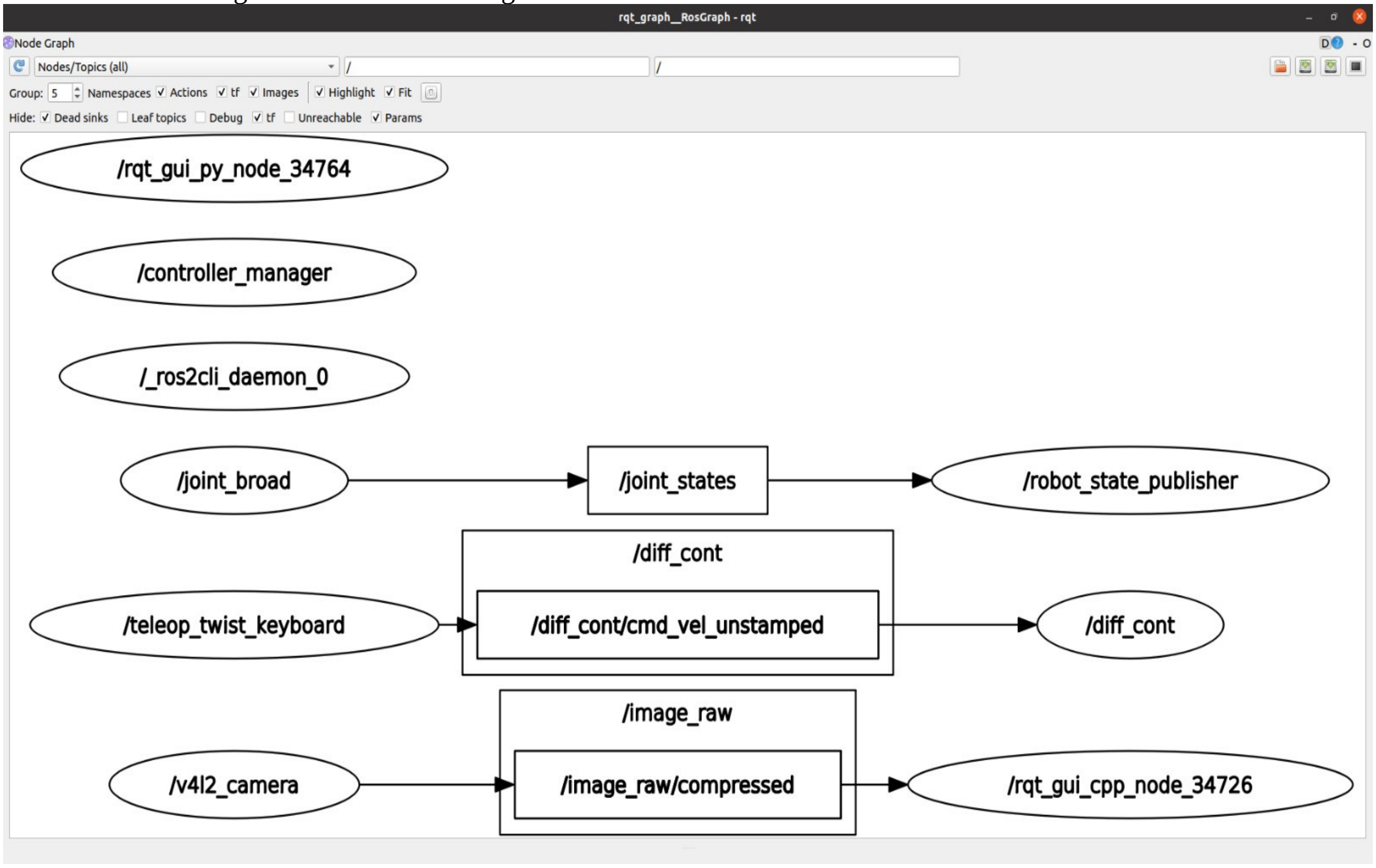
Figure 4.8: ROS2 Topic list.



```
        D.size() = 0, expected 5
[INFO] [1675429797.711359310] [image_saver]: Saved image left0014.jpg
[ERROR] [1675429797.711537599] [camera_calibration_parsers]: Videre INI format can only
save calibrations using the plumb bob distortion model. Use the YAML format instead.
        distortion_model = '', expected 'plumb_bob'
        D.size() = 0, expected 5
^C[INFO] [1675429797.894390426] [rclcpp]: signal_handler(signal_value=2)
royyan@royyan-IdeaPad-Gaming-3-15IHU6:~/bags/test2$ cd ../..
royyan@royyan-IdeaPad-Gaming-3-15IHU6:~$ ros2 topic list
/camera_info
/diff_cont/cmd_vel_unstamped
/dynamic_joint_states
/image_raw
/image_raw/compressed
/image_raw/compressedDepth
/image_raw/theora
/joint_states
/odom
/parameter_events
/robot_description
/rosout
/tf
/tf_static
royyan@royyan-IdeaPad-Gaming-3-15IHU6:~$ 
```

A graph of the interaction between ROS nodes can then be produced using rqt_graph, this is a built in analytics tool in ROS to troubleshoot the robot. Figure 4.9 illustrates these interactions.

Figure 4.9: ROS2 Running nodes.



rqt_graph__RosGraph - rqt

Node Graph

Nodes/Topics (all) / /

Group: 5 Namespaces ✓ Actions ✓ tf ✓ Images ✓ Highlight ✓ Fit

Hide: ✓ Dead sinks ☐ Leaf topics ☐ Debug ✓ tf ☐ Unreachable ✓ Params

Chapter 5 Conclusion

The objective of this study was to 1) Develop Unmanned Ground Vehicle for photogrammetry application. 2) Evaluate the quality of 3D mesh models generated using UGV.

It can be concluded that a ROS2 compatible mobile robot has been fabricated, and can be controlled wirelessly via a computer on the same network. The robot is capable of taking pictures and automatically storing it onto the computer on the same network without manually connecting the raspberry pi SD card into the computer. The resulting quality of the 3D model is satisfactory for single camera with relatively low resolution, it is better than the photogrammetry rover design by Olaf Hallan Graven (2018), however worse than specialised museum photogrammetry with DSLR camera. Nevertheless, the result was a huge improvement from preliminary results.

5.1 Considerations

- A major flaw with this mobile robot is its camera quality limitation due to processing power, while 720p is adequate for an inspection mobile robot, it is not the best for photogrammetry.
- Odometry worked in simulation mode, however did not in the real robot because the motors used lacked encoders seen in more expensive units, getting this to work will allow for implementation of Nav2 and therefore partial autonomous navigation.
- The chasis of the robot would not support the weight of 2D LiDAR, thus a modified chasis will have to be fabricated for further developments of autonomout navigaton.

5.2 Significance

In this study, the feasablility of photogrammetry using a small mobile robot has been explored. The outcome of this project has provided a framework to produce low-cost ROS2 compatible mobile robots. Currently the most popular training tool for ROS is the turtlebot3, this includes a camera and 2D lidar, however costs 8000 MYR, a significant barrier to entry for students to start out. This project will allow future students to test computervision algorithms on mobile robots without having to aquire a turtlebot.

REFERENCES

[1] Chen, M.-C. and Huang, S.-H. (2003) ‘Credit scoring and rejected instances reassigning through evolutionary computation techniques’, *Expert Systems with Applications*, 24(4), pp. 433–441.

[2] T.B. Asafa, T.M. Afonja, E.A. Olaniyan, H.O. Alade (2016) ‘Development of a vacuum cleaner robot’, Mechanical Engineering Department, Ladoke Akintola University of Technology

[3] Olaf Hallan Graven (2018) ‘An Autonomous Indoor Exploration Robot Rover and 3D Modeling with Photogrammetry’, University College of Southeast Norway

[4] M. Z. H. Noor, S. A. S. M. Zain, L. Mazalan (2013) ‘Design and Development of Remote-Operated MultiDirection Unmanned Ground Vehicle (UGV)’, Faculty of Electrical Engineering Universiti Teknologi Mara

[5] Abhishek Roya , Mathew Mithra Noel (2014) ‘Design of a high-speed line following robot that smoothly follows tight curves’, Bosch Engineering and Business Solutions, Bangalore

[6] Joshua J. Medina, James M. Maley, Siddharth Sannapareddy, Noah N. Medina, Cyril M. Gilman, John E. McCormack (2020) ‘A rapid and cost-effective pipeline for digitization of museum specimens with 3D photogrammetry’, Moore Laboratory of Zoology, Occidental College, Los Angeles, CA

[7] Adrian Savari Thomas, Mohd Fahrul Hassan1, Al Emran Ismail1, Reazul Haq Abdul Haq, Ahmad Mubarak Tajul Arifin1, Md Fauzi Ahmad (2019) ‘Portable Mini Turntable for Close-Range Photogrammetry: A Preliminary Study’, Faculty of Mechanical and Manufacturing Engineering, Universiti Tun Hussein Onn Malaysia

APPENDIX A

ROBOT_CORE CODE:

```
<?XML VERSION="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" >

<xacro:include filename="inertial_macros.xacro"/>

<material name="white">
<color rgba="1 1 1 1" />
</material>

<material name="orange">
<color rgba="1 0.3 0.1 1"/>
</material>

<material name="blue">
<color rgba="0.2 0.2 1 1"/>
</material>

<material name="black">
<color rgba="0 0 0 1"/>
</material>

<!-- BASE LINK -->

<link name="base_link">

</link>
<!-- CHASSIS LINK -->

<joint name="chassis_joint" type="fixed">
<parent link="base_link"/>
<child link="chassis"/>
<origin xyz="-0.1 0 0"/>
</joint>

<link name="chassis">
<visual>
<origin xyz="0.15 0 0.075"/>
<geometry>
<box size="0.3 0.3 0.15"/>
</geometry>
<material name="white"/>
</visual>
```

```
<collision>
<origin xyz="0.15 0 0.075"/>
<geometry>
<box size="0.3 0.3 0.15"/>
</geometry>
</collision>
<xacro:inertial_box mass="0.5" x="0.3" y="0.3" z="0.15">
<origin xyz="0.15 0 0.075" rpy="0 0 0"/>
</xacro:inertial_box>
</link>
<gazebo reference="chassis">
<material>Gazebo/White</material>
</gazebo>
<!-- LEFT WHEEL LINK -->

<joint name="left_wheel_joint" type="continuous">
<parent link="base_link"/>
<child link="left_wheel"/>
<origin xyz="0 0.175 0" rpy="-${pi/2} 0 0" />
<axis xyz="0 0 1"/>
</joint>

<link name="left_wheel">
<visual>
<geometry>
<cylinder radius="0.05" length="0.04"/>
</geometry>
<material name="blue"/>
</visual>
<collision>
<geometry>
<cylinder radius="0.05" length="0.04"/>
</geometry>
</collision>
<xacro:inertial_cylinder mass="0.1" length="0.04" radius="0.05">
<origin xyz="0 0 0" rpy="0 0 0"/>
</xacro:inertial_cylinder>
</link>
<gazebo reference = "left_wheel">
<material>Gazebo/Blue</material>

</gazebo>

<!-- RIGHT WHEEL LINK -->

<joint name="right_wheel_joint" type="continuous">
<parent link="base_link"/>
<child link="right_wheel"/>
<origin xyz="0 -0.175 0" rpy="${pi/2} 0 0" />
```

```
<axis xyz="0 0 -1"/>
</joint>

<link name="right_wheel">
<visual>
<geometry>
<cylinder radius="0.05" length="0.04"/>
</geometry>
<material name="blue"/>
</visual>
<collision>
<geometry>
<cylinder radius="0.05" length="0.04"/>
</geometry>
</collision>
<xacro:inertial_cylinder mass="0.1" length="0.04" radius="0.05">
<origin xyz="0 0 0" rpy="0 0 0"/>
</xacro:inertial_cylinder>
</link>
<gazebo reference = "right_wheel">
<material>Gazebo/Blue</material>

</gazebo>
<!-- CASTER WHEEL LINK -->

<joint name="caster_wheel_joint" type="fixed">
<parent link="chassis"/>
<child link="caster_wheel"/>
<origin xyz="0.24 0 0"/>
</joint>

<link name="caster_wheel">
<visual>
<geometry>
<sphere radius="0.05"/>
</geometry>
<material name="black"/>
</visual>
<collision>
<geometry>
<sphere radius="0.05"/>
</geometry>
</collision>
<xacro:inertial_sphere mass="0.1" radius="0.05">
<origin xyz="0 0 0" rpy="0 0 0"/>
</xacro:inertial_sphere>
</link>
<gazebo reference = "caster_wheel">
<material>Gazebo/Black</material>
```

```
<mu1 value = "0.001"/>
<mu2 value = "0.001"/>


</gazebo>
</robot>
```

APPENDIX B

ROBOT URDF CODE

```
<?XML VERSION="1.0"?>

<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="robot">
<xacro:include filename = "robot_core.xacro"/>
<xacro:include filename="ros2_control.xacro" />
<xacro:include filename="camera.xacro" />
</robot>
```

ROS2_CONTROL CODE

```
<?XML VERSION="1.0"?>

<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
<ros2_control name="GazeboSystem" type="system">
<hardware>
<plugin>gazebo_ros2_control/GazeboSystem</plugin>
</hardware>
<joint name="left_wheel_joint">
<command_interface name="velocity">
<param name="min">-10</param>
<param name="max">10</param>
</command_interface>
<state_interface name="velocity"/>
<state_interface name="position"/>
</joint>
<joint name="right_wheel_joint">
<command_interface name="velocity">
<param name="min">-10</param>
<param name="max">10</param>
</command_interface>
<state_interface name="velocity"/>
<state_interface name="position"/>
</joint>
</ros2_control>

<gazebo>
<plugin name="gazebo_ros2_control"
filename="libgazebo_ros2_control.so">
<parameters>$(find my_bot)/config/my_controllers.yaml</parameters>
</plugin>
</gazebo>

</robot>
```

APPENDIX C

LAUNCH FILE

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import
PythonLaunchDescriptionSource
from launch_ros.actions import Node
def generate_launch_description():
package_name='my_bot'

rsp = IncludeLaunchDescription(
PythonLaunchDescriptionSource([os.path.join(
get_package_share_directory(package_name),'launch','rsp.launch.py'
)]), launch_arguments={'use_sim_time': 'true'}.items()
)

# Include the Gazebo launch file, provided by the gazebo_ros package
gazebo = IncludeLaunchDescription(
PythonLaunchDescriptionSource([os.path.join(
get_package_share_directory('gazebo_ros'), 'launch', 'gazebo.launch.py')]),
)
spawn_entity = Node(package='gazebo_ros', executable='spawn_entity.py',
arguments=['-topic', 'robot_description',
'-entity', 'my_bot'],
output='screen')

diff_drive_spawner = Node(
package="controller_manager",
executable="spawner.py",
arguments=["diff_cont"],
)
joint_broad_spawner = Node(
package="controller_manager",
executable="spawner.py",
arguments=["joint_broad"],
)
return LaunchDescription([
rsp,
gazebo,
spawn_entity,
diff_drive_spawner,
joint_broad_spawner ])
```

ABSTRACT

The purpose of this study is to investigate and develop a novel, and cost effective photogrammetric scanning methodology, suitable for reconstructing a full 3D digital model. Robotics has become very popular in this era, with high precision and complex functionality. One novel industrial application in robotics is its use to create 3D point cloud data on infrastructure to inspect any defects in need of maintenance, this is normally done with 3D LiDAR sensors. However, cameras are also capable of creating point cloud data with textures, using a method known as photogrammetry. Hence, the aim of this project is to develop a teleoperated (master-slave) mobile robot that is capable of taking image data and sending the information automatically to another compter. The UGV circles around the object at fixed angles, the movement path is in the shape of a circle that rotates at fixed angles, while the digital camera, mounted above the robot frame, captures the images. The image is then processed into point cloud and 3D mesh with texture. In this paper, we implemented ROS2 control framework and v4l2 capture interface on a differential drive robot and USB webcam. The mobile robot uses an existing chassis design on the market. The experimental results show that our system was able to perform teleoperation and photogrammetry of target objects in controlled lighting condition.