MAKERNOVA

DRISHTI
A Revolutionary Concept

TELE & AUTO - OPERATED BOT

Problem Statement:

To enable remote control and interaction with a robot in the Gazebo ROS framework, both in simulation and physical hardware, using keyboard input.

Outcome:

1. Implemented and simulated Goal-To-Goal algorithm in Turtlesim.
2. Programmed autonomous algorithm and implemented on hardware using ultrasonic sensors.

Team Members:

Adarsh Kumar Srivastava

Akhil Rai

Dipsi Hadiya

Hritick Kumar

Jashwanth Simha

Jeet Ariwala

Jiya Parmar

Krutika Joshi

N.Sanjana

Naman Omar

Neem Sheth

Sahil Umadiya

Saurabh Mishra

Shivansh Tyagi

Mentors:

Anant Agrawal

Devam Shah

Kriya Jain

INDEX

Dual Booting

- Dual booting is a way of using two or more different operating systems (OS) on a single computer. Typically, each operating system is installed on a separate "partition" on the main hard drive.
- Virtual Machine is a computer system created using software on one physical computer in order to emulate the functionality of another separate physical computer.
- Linux is known for its stability, which is a critical factor for servers that need to be up and running 24/7. Linux systems are designed to be highly reliable, with fewer crashes and downtime compared to Windows. Hence, due to these reasons we installed Ubuntu 20.04 for our project.
- Some important steps that were involved during the dual booting process are:
    - Creation of **unallocated space** for Ubuntu installation.
    - Installation and use of **rufus 4.1** to flash the USB Drive and make it bootable.
    - Installation of **Ubuntu 20.04** and providing specified space to each partition.

ROS Installation And Terminologies

- ROS stands for Robot Operating System. It is a set of software libraries and tools that help you build robot applications.
- The steps undertaken for the utilization of **ROS NOETIC**:
    - Setting up source.list
    - Setting up keys
    - Installation
    - Environment setup:
    - Dependencies for building packages:
    - Initialise rosdep
- The Robot Operating System (ROS) is a **set of software libraries and tools that is freely available due to its open-source nature** that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project.
- The key features are Interoperability, Modularity & Flexibility, Robustness and Portability.
- **Node** (small programmable unit): A node is an executable file within a ROS package. Nodes can publish or subscribe to a Topic.
- **Master** (connection b/w the nodes): It tracks publishers and subscribers to topics as well as services. The role of the master is to enable individual ROS nodes to locate one another.
- **Topic:** The named buses over which nodes exchange data. It receives the same type of data from the publisher which is used by subscribers.
- **Package:** A ROS package is simply a directory descended from ROS_PACKAGE_ that has a package.xml file in it. Packages are the most atomic unit of build and the unit of release. This means that a package is the smallest individual thing you can build in ROS and it is the way software is bundled for release (meaning, for example, there is one debian package for each ROS package), respectively.
- **Workspace:** It typically refers to the directory structure and environment where you develop, build, and manage your ROS packages and code. The workspace is where you organize your projects, libraries, and dependencies for developing robotic applications.

ROS Commands

- roscore
    - Syntax: roscore
    - roscore is a collection of nodes and programs that are pre-requisites of a ROS-based system.
    - You **must** have a roscore running for ROS nodes to communicate. It is launched using the roscore command.
- rosnode
    - Syntax: rosnode info [node name]
    - rosnode is a command-line tool for displaying debug information about ROS nodes, including publications, subscriptions, and connections. It also contains an experimental library for retrieving node information.
- rosrun
    - Syntax: rosrun [package name] [executable name]
    - rosrun allows you to use the package name to directly run a node within a package (without having to know the package path).
- rostopic
    - Syntax: rostopic <subcommand> <topic name>
    - rostopic contains the rostopic command-line tool for displaying debug information about ROS topics, including publishers, subscribers, publishing rate, and ROS messages.
- roslaunch
    - Syntax: rosrun [package name] [launch file name]
    - roslaunch is a tool for easily launching multiple ROS nodes locally and remotely via SSH.

Publisher And Subscriber Files

- Publisher File: It is the sender file. It packs the data and sends them to specific topics.
- Subscriber File: It is the receiver file. It unpacks the received data from the specific topic.
- For our project, we required multiple subscribers and publishers to send and receive the data for various components.
- And hence, we were required to make a single ROS file which could work as both publisher and subscriber file.
- So, a single file could be able to receive the data then make the required changes and process the data accordingly and then send that data over another topic.
- In Our final project file, we made two Rosserial file (explained later in this documentation) (basically an Arduino linked ROS file) which worked as our publisher and subscriber simultaneously.

- TCPROS and UDPROS: ROS typically relies on TCPROS (Transmission Control Protocol for ROS) and UDPROS (User Datagram Protocol for ROS) protocols for communication between nodes. These protocols define the way messages are serialized, transmitted, and deserialized between nodes.

TurtleSim

- turtlesim is a tool made for teaching ROS and ROS packages. Turtlesim is a lightweight simulator (GUI) for learning ROS.
- How to start turtlesim in terminal :
    - roscore (the Master must run to allow the nodes to communicate)
    - rosrun turtlesim turtlesim_node (Run turtlesim)
- The /turtlesim node publishes two topics:
    - /turtle1/pose with the message type [turtlesim/Pose]
    - /turtle1/cmd_vel have the message type [Twist ()]
- Twist:
    - “twist" typically refers to a specific type of message used to represent linear and angular velocities of a robot or object in a 2D, or 3D space.
    - It's particularly useful for controlling robots in a holonomic or non-holonomic manner, allowing them to move in any direction or follow specific trajectories.
    - By publishing a geometry_msgs/Twist message to the appropriate topic, you can control the movement of a robot.
- Pose:
    - A "pose" refers to the position and orientation of an object in space.
    - A pose is typically represented using a combination of coordinates and angles that describe the object's location and orientation relative to a reference frame.
    - The concept of poses is fundamental in robotics for tasks such as localization, navigation, and control.
- For preparing for the project, we performed various tasks with the aid of turtlesim which involved moving the turtle in circle and moving our turtle in square or rectangular path.

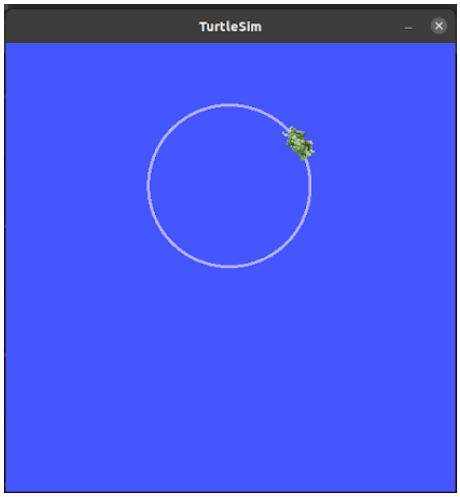

Fig 1: Turtle movement in circle

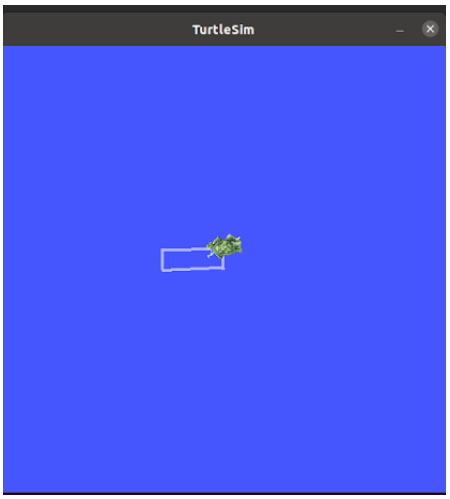

Fig 2: Turtle movement in rectangular path

Rosserial Arduino

- Rosserial is a protocol for wrapping standard ROS serialized messages and multiplexing multiple topics and services over a character device such as a serial port or network socket. Client libraries allow users to easily get ROS nodes up and running on various systems. These clients are ports of the general C++ roserial_client library.
- It provides a way to interface with low-level hardware, such as sensors, actuators, and other devices, using the ROS ecosystem.
- Rosserial allows you to integrate robotic components with different levels of computational capability into a larger ROS-based system.
- Using rosserial to setup a ROS node on an ARDUINO:
    - The rosserial package can allow a computer running ROS to communicate with a node on a microcontroller via ROS topics that it can subscribe and publish.
    - Enables the heavy programs on a device and sends any needed information to the microcontroller so that it can drive whatever hardware robot is using.
    - rosserial is a protocol to send data through a serial interface. In a client-server rosserial implementation, a rosserial-server is a computer running ROS and a rosserial-client is the microprocessor that receives sensors' data and transports it to the server in the form of ROS messages. rosserial-server in this implementation is a publishing node while rosserial-client is a subscriber node.

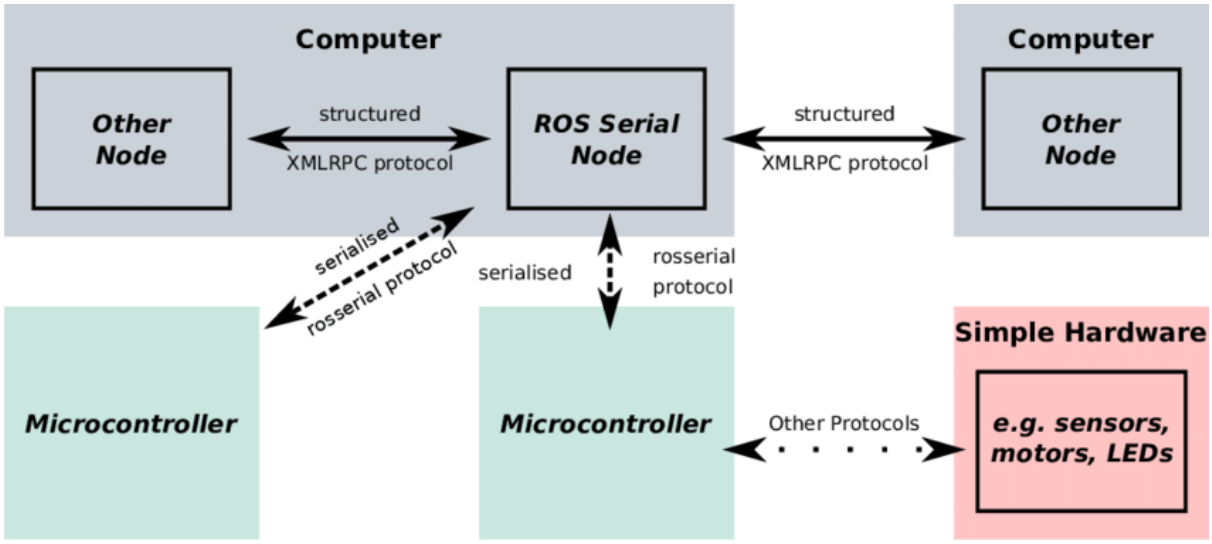

Fig 3: rosserial communication flowchart

Hardware

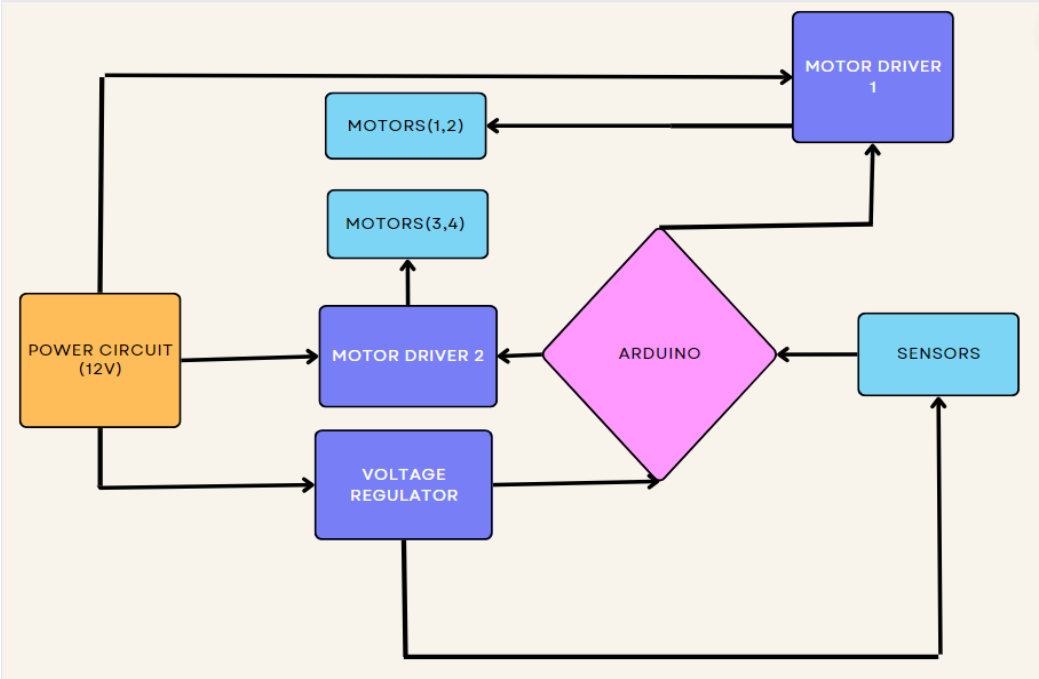

Fig 4: Flowchart For hardware

- Ultrasonic Sensor (HCSR04)
  - Ultrasonic sensors are widely used in various applications to measure distances and detect objects. These sensors work by emitting high-frequency sound waves and measuring the time it takes for the waves to bounce back after hitting an object.
  - In our Project, 3 ultrasonic sensors were used, each mounted having 60 degrees of angle between each of them to measure the distance efficiently.
  - Trig (Trigger) pin is used to trigger the ultrasonic sound pulses.
  - The trigger is a control signal that you send to the ultrasonic sensor to initiate the measurement. When the sensor receives the trigger signal, it emits a burst of ultrasonic waves. This burst travels towards the target object.
  - The echo is the reflected sound wave that returns to the sensor after bouncing off the target object. The ultrasonic sensor detects this echo and measures the time it takes for the sound wave to travel to the object and back. Using the speed of sound, the sensor can then calculate the distance to the object.]
  - Connection with Arduino UNO:

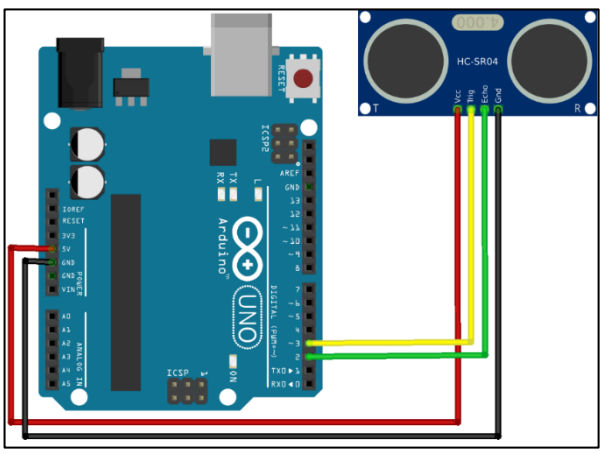

Fig 5: Connection of the Ultrasonic sensor with Arduino board

- Cytron Motor drivers
    - It is a motor driver module or board developed by Cytron.
    - It is used to control the speed and direction of Motors in various applications such as Robotics, Automation etc. and is compatible with microcontrollers such as Arduino.
    - It has 3 pins, one for ground, one for speed (PWM Pin) and one for its direction (Digital Pin).
    - Connection with Arduino UNO:

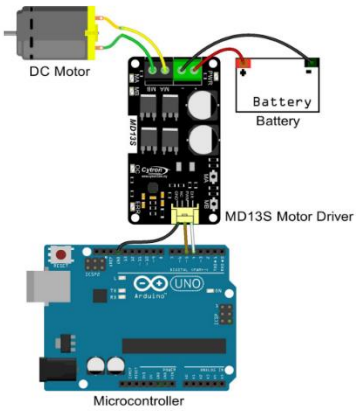

Fig 6: Connection of Motor driver with Arduino Board

- Arduino Uno
    - Arduino UNO is a microcontroller board based on the **ATmega328P** (provides UART (Universal Asynchronous Receiver-Transmitter) protocol). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz resonator, a USB connection, a power jack, and ATmega16U2 (USB-to-Serial Converter).
    - For our project, we have used 2 Arduino Uno boards to minimise the load of the software.
    - Tech Specifications:

Microcontroller	ATmega328P
Operating Voltage	5v
Digital I/O Pins	14(of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20mA
DC Currnet for 3.3V pin	50mA
Flash memory	32KB(Atmega328P)of which 0.5KB used by bootloader
SRAM	2KB(ATmega328P)
EEPROM	1KB(ATmega328P)
Clock speed	16MHz
LED BUILTIN	13
Length	68.6mm
Width	53.4mm
Weight	25g

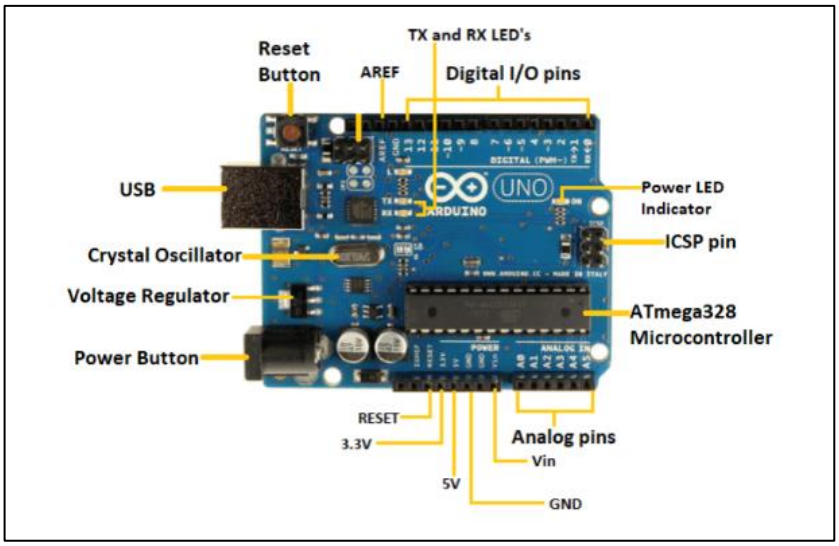

Fig 7: Arduino UNO Board

- Uses of the components in our circuits:
  - Ultrasonic Sensors were used to measure distance and publish to Arduino for further calculations of direction, speed and angle.
  - Motor Drivers were used to control speed and direction (either clockwise or anticlockwise) of motors.
  - Arduino UNO board and the IDE was used for accepting the ultrasonic data that data is being published. Later, the IDE subscribes the data and sends to the motor driver.
  - Three parallel screw connectors were connected to the LiPo battery. Out of which, 2 of them were connected to motor drivers for 12 volts of supply. And the remaining one was connected to the main circuit of the bot.
  - Voltage regulator was used to get the voltage drop up to 5 volts from the 12 volts battery.

Teleoperation In Turtlesim

- A ROS mobile application is a software application designed to run on a mobile device, such as a smartphone or tablet, and interact with the ROS (Robot Operating System) ecosystem.
- The purpose of a ROS mobile application is to provide a user interface that allows remote control, monitoring, and interaction with robots or robotic systems that are integrated within the ROS framework. This type of application enables users to manage and operate robots remotely, view sensor data, send commands, and receive feedback through a mobile device.
- As a part of our task, we used the joystick from the ROS mobile application for tele operational movement of our turtle in the turtlesim.
- The joystick data is being published on the /turtle1/cmd_vel and the turtlesim GUI subscribes that data and moves accordingly.

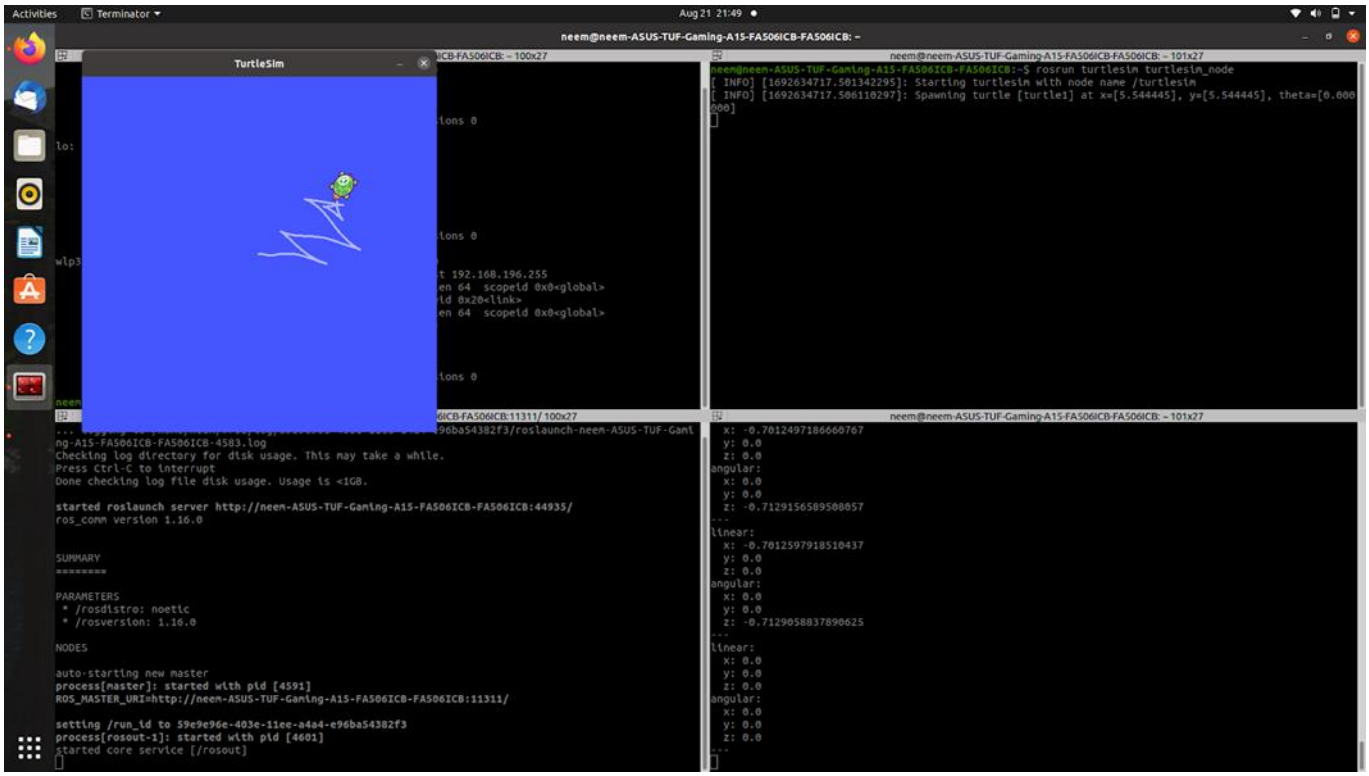

Fig 8: Teleoperation of turtle

Goal To Goal

- Goal to Goal refers to a navigation scenario where a virtual turtle is tasked with moving from one specified goal point to another in the simulated 2D environment provided by the turtlesim node.
- We had designed the algorithm for this task with the help of Twist () and Pose () message type which provided velocities and position respectively.
- The flow for this task went with asking the user the desired coordinates. Then the ROS file subscribes the position of the turtle currently.
- Then, the appropriate calculations are made for the angle at which it must move and at what distance. This is now being provided to the turtle continuously.
- When we publish the linear and angular velocities simultaneously, our turtle moves towards the goal with a smooth turning path because we are providing the speeds dependent on its real-time position.
- We learnt that, keeping the hardware and physical aspects in mind for the bot, we need to scale our velocities such that our objective is achieved in most effective way.

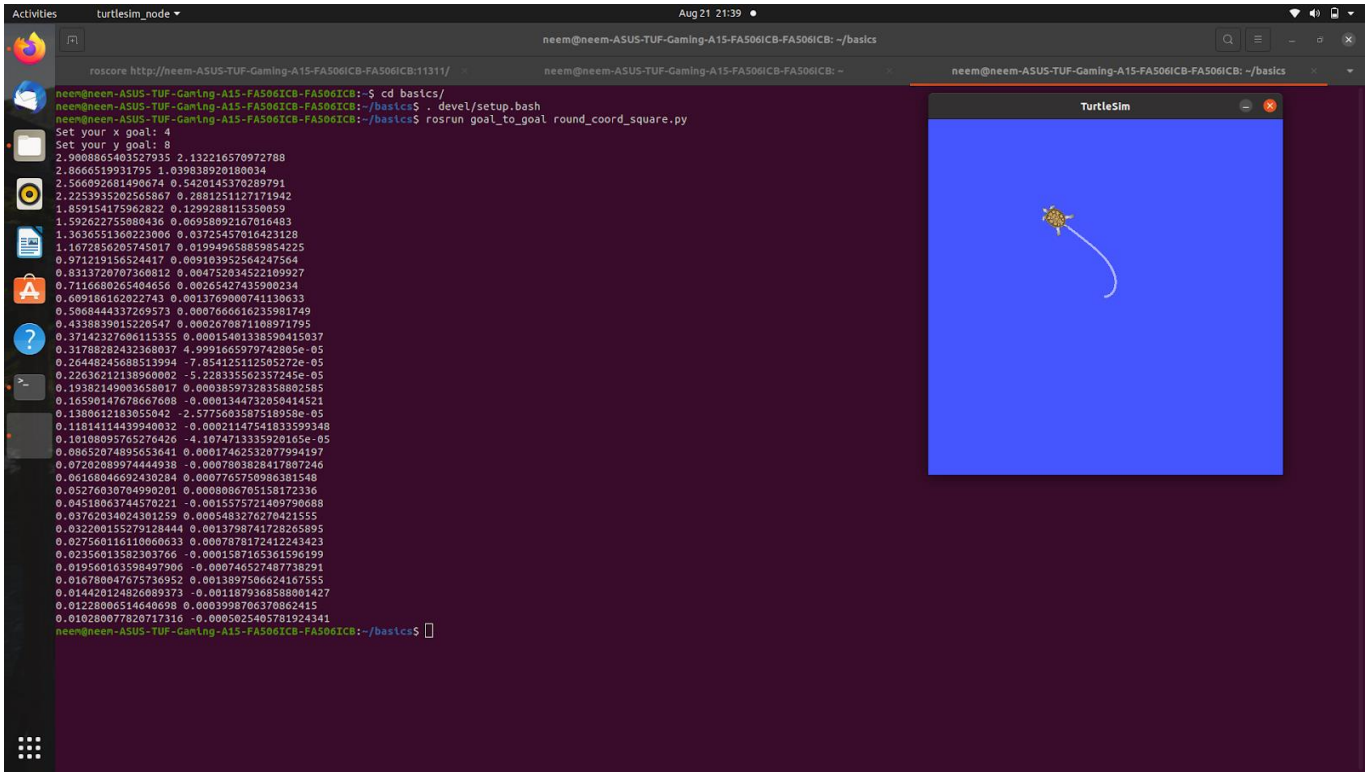

Fig 9: Goal to Goal Implementation

PID Based Autonomous Algorithms

- PID stands for Proportional, Integration and Differentiation. Here we had developed this algorithm with the aim to have a smooth avoidance mechanism, small avoidance curvature and minimising the accumulation of error.
    - Proportionality specifies the steering angle for the bot.
    - Differentiation specifies the perpendicular speed and minimises the errors.
    - Integration specifies the original path to be followed after obstacle avoidance.
- We have created a Rosserial-Arduino file which connects first Arduino board and the ultrasonic sensors. This file subscribes the distance of the bot from obstacle and publishes the same.
- The ROS file subscribes to the data, the error is calculated and PID calculations are implemented.
- The Equation of PID calculation is as follows:
    - Speed = K_p*P + K_I*I + K_D*D
    - P = Error measured in ultrasonic sensor
    - I = Integration of Previous Errors (Last 10 Errors with the help queue)
    - D = Differentiation of Last Two Recent Errors
    - K_p, K_I and K_D are three tuning constants which are set according to our hardware implementation.
- The Ultrasonic data from left and right sensors are processed for PID speeds and then these speeds are given to the motor drivers accordingly.
- The direction of the wheels i.e., forward, or backward is decided on the basis of of three cases which are as follows:
    - When the obstacle is too near, the bot moved in a curved reverse direction. Both wheels in backward direction.
    - When an obstacle is at intermediate distance or exactly in front, the bot makes a sharp right turn. Right wheel in backwards direction and left wheel in forward direction.
    - In all the other cases the bot has stabilised, distance dependent velocity and direction. Both the wheels in forward direction.
- These calculated speeds are published and subscribed on another Rosserial-Arduino file which is connects second Arduino board and the motor drivers.
- This file subscribes the speed, and this speed is scaled according to analog output to be sent over PWM pins and the direction-bit is sent as digital output.

Vector Based Autonomous Algorithm

- The main idea of this algorithm is used to travel in the resultant direction where the obstacle isn't present.
- We have created a Rosserial-Arduino file which connects first Arduino board and the ultrasonic sensors. This file subscribes the distance of the bot from obstacle and publishes the same.
- In the ROS File, while rospy is not shut down and till the sensor detects the distance from any of the sensor run the below steps:
    - Calculate the resultant distance and resultant angle by considering the left distance as –(x-axis) vector, right distance as +(x-axis) vector and middle distance as +(y-axis) vector.
    - So, the bot must move on the resultant direction, in order make that turn give the left velocity proportional to = (180-angle)
    - Right velocity directly proportional to = angle
    - Velocity1 = (180-angle) *distance*k
    - Velocity2 = (angle)*distance*k
- These calculated speeds are published and subscribed on another Rosserial-Arduino file which is connects second Arduino board and the motor drivers.

Software Errors Encountered:

- Error- usb drive not detected in rufus.
- Correction- disabling protegent antivirus for time being for rufus to detect the drive.

- Error - gpg: no valid OpenPGP data found.
- Correction: Using LAN-WIFI for installation process instead of hotspot.

- Error: Failed to connect to master
- Correction: Run roscore command always before running any other ROS commands

- Error - Bad interpreter: No Such file or directory
- Correction: Modification in CmakeList.txt and package.xml file by uncommenting and adding of dependencies

- Error- package not found
- Correction- . devel/setup.bash
- For adding several environment variables which ROS needs in order to work

Hardware Errors Encountered:

- Wiring Error: Usage of jumper wire had led to complex circuit design and difficulty in analysis.
- Solution: Usage of header wires and soldering of the circuit to simplify the wiring.

- Excess Load on Single Microcontroller: The use of arduino uno for both motor driver and sensors lead to performance failure due to heavy load.
- Solution: Use two microcontrollers (2 arduino Uno) or using one arduino Uno plus one arduino mega, since mega has 8 times more memory as compared to uno so we divide the sensors in one board and motor drivers in another board.

Task Videos Links:

- Goal To Goal Video: https://drive.google.com/file/d/1NxBZgexsc7T_Tx3PBwPOSh8pTrOKqnZc/view?usp=sharing
- Tele-Operation Turtlesim Video: https://drive.google.com/file/d/1LP6sOAWH9jR7rrkJRrkNt0iVSU6e_LmT/view?usp=sharing
- Tele-Operation Bot Video: https://drive.google.com/file/d/1LKK0j20h09Irj0lkQfX9_Li5E579cmmQ/view?usp=sharing

GitHub Repository Link:

- https://github.com/Neem-Sheth/Tele-And-Auto-Operated-Bot.git

Bibliography

- https://store.arduino.cc/products/arduino-uno-rev3
- https://www.ros.org/
- https://ubuntu.com/robotics/what-is-ros
- https://subscription.packtpub.com/book/iot-and-hardware/9781788479592/1/ch01lvl1sec15/ros-commands-summary