ERC Hackathon 2024

1 General Overview

Welcome to the **ROS2 Robotics Hackathon: Campus Power Quest!**

Problem Statement:

A sudden electrical outage has plunged BITS Goa into darkness, and no immediate remedy is in sight. It is now up to us, the students of the Electronics and Robotics Club (ERC), to **'bring back the lite'**. Your mission, should you choose to accept it, is to design an advanced electrical surveillance robot capable of navigating the campus, analyzing power machinery, and identifying regions of electrical failure.

This hackathon aims to introduce you to the broad domains that form a robotic system, namely Design (Mechanical and Electronics) and Automation (Perception, Planning, and Control), and guide you in building your own robotic system from scratch. Each aspect of the task is crucial to achieving the final goal, yet they can be tackled independently. Your solution should encompass the mechanical design, electronics interfacing, and an implementation of autonomous behavior.

Don't Panic!

At first glance, the problem statement may seem daunting. But fear not! It is intentionally complex and challenging, designed to stretch your abilities and creativity. Remember, the key is the thought and effort that goes into your attempt, so be sure to document everything meticulously!

We hope you will gain valuable insights and skills from this hackathon. The learning resources section of this document is a great starting point, but we encourage you to explore and search online as much as possible.

Need Help?

If you encounter difficulties during the hackathon or have any doubts, don't hesitate to reach out for help. However, the best method to learn is through practice, so we encourage you to debug errors yourself as much as possible.

Ready to accept the challenge and light up our campus?

Fill out this form to register for the hackathon and embark on an electrifying adventure!

[Link to the Github Repository.]

2 Logistics

2.1 Submission Guidelines

The hackathon is meant to be a group activity, with groups consisting of a maximum 6 members (ideally 2 for each subsystem). We will allot teams to those who want to participate but haven't formed a team. One of the team members should create a GitHub Repository consisting of all the required documents, files and code that the team wants to submit. It should be properly organized with the README.md file consisting of thorough documentation of your attempt. Also include the required CAD files, screenshots of your robot design, link(s) to electronics simulations, a screen recording of your robot performing the navigation as well as a rosbag in the repository.

You will be assigned mentors for each vertical who will help you in case of any queries and have general discussions about your approach. More details about this will be given on the groups.

Fun and creative team names will be rewarded :P

Submission Deadline: 1st August (Tentatively)

2.2 Contact Information

Automation: Ritwik Sharma
Vimarsh Shah
Ajinkya Deshpande
Sharvil Potdar
Garv Gupta
Nayan Gogari

Electronics: Parth Shah
Ayush Tiwari
Nilay Wani
Atul M

Mechanical: Ansh Parmeshwar
Harsh Jain
Hritik Joglekar
Snigdha Semwal

3 Learning Resources

The ERC Handbook is an extensive compilation of resources related to robotics so do check it out. The resources below can be used in addition to the handbook:

3.1 Python

Check out these video series (1, 2) by Cody Schafer. They cover the basics of python from variables to functions and even object oriented programming. This guide by Google is also a good supple- mentary resource.

3.2 Linux Terminal

You should do your best to familiarize yourself with the Linux terminal as it's essential for many of the development tools that you will use. Get started here and here.

3.3 Git

An integral part of most software projects, Git is a tool for version control and managing changes to code. Check out this course and the git cheat sheet.

3.4 ROS2

The Robotics Operating System is the backbone of every complex robotics stack. You will need to first set it up on your machine. For this hackathon, we require you to install **ROS2 Humble** on your systems. Since ROS 2 is supported for Linux only, we recommend that you dual boot with Ubuntu 22.04. If this isn't possible, you can try either a virtual machine or Windows Subsystem for Linux for Windows users. To use a GUI application like Gazebo with WSL you'll need to set up up xserver. For mac users, follow this video to setup Ubuntu (remember to install Ubuntu 22.04 only). After getting to a Linux command prompt, follow the ROS2 Humble installation through this doc for instructions. Detailed Instructions here.

The best learning resource for ROS is the official ROS2 Humble tutorials. Go through the tutorials thoroughly up till 'creating custom msg and srv files'. You are also free to refer any other resources or video series for the same

3.5 Path Planning

The following doc has several resources you could use to learn different path planning algorithms. If you want a more mathematically rigorous resource you can go through this book. You don't need to use shapely for obstacle avoidance, we have provided you with the functions to do so.

3.6 Controls

Robotics deals with many continuously operating, dynamic systems, which are dealt with using Control Theory tools. For the basic concepts, check out the course on controls of a mobile robot by GATech and this video series by Steve Brunton. The PID controller is the most commonly used con- troller. Check out this introductory series by MATLAB. The PID controller section of our handbook provides more mathematical insight as well. Note that going above and beyond to implement different control algorithms will fetch extra credit ;)

3.7 Computer Vision

You can follow this playlist by Sentdex for learning OpenCV (till video 13). Refer to the OpenCV website for or this page for more tutorials.

3.8 Mechanical

A robust and optimized mechanical design is the backbone of every deployed robotic system. Get started with an introduction from ERC's handbook. Refer to this playlist to get started with CAD in Fusion 360. ANSYS is a simulation software which you can use to perform structural analysis. Use this to learn more.

3.9 Electronics

Have a look at the ERC handbook's page on Arduino here. Jeremy Blum's playlist is a great place to get started. Arduino's official documentation is also well made and useful. Use KICAD for designing your PCBs. Refer to these videos to learn about PCB design using KICAD.

4 Task Details

Your team needs to do all three tasks to be considered a complete submission. Different team members can work on different tasks, or on multiple tasks. The goal of the hackathon is to introduce everyone to the basics of all three domains and to give a taste of how system design feels like. Focus on learning and implementation of what you have learnt rather than the competition, although the winning team will be given a treat on campus :)

4.1 Mechanical

Objective:

Design a robust, reliable, and efficient mechanical structure for an autonomous electrical surveillance robot capable of navigating the BITS Goa campus, including climbing stairs and traversing uneven terrain. The robot must transport spare parts for fixing electrical failures and use a robotic arm for inspection and repairs.

Specifications:

1. **Chassis Design:**
    - **Dimensions:** As per your requirements (should be able to justify your choice).
    - **Mobility:** Wheeled or continuous track (chain and sprocket mechanism) mechanism for stair climbing and terrain traversal.
    - **Ground Clearance:** Ensure sufficient clearance for navigating uneven terrains.
    - **Compartments:** Design compartments for storing batteries, electronic components, and spare parts.
    - **Camera Mounts:** Include a front camera mount inside the chassis and an elevated camera mount for surrounding inspection.
2. **Suspension System:**
    - Implement a suspension system to absorb shocks and vibrations while traversing various terrains.
    - The tracks or wheels or any other mechanism used should be suitable for multiple terrains.
3. **Robotic Arm:**
    - **Design:** Create a 2/3 DoF (degrees of freedom) robotic arm with a gripper.
    - **Gripper:** The gripper has to perform tasks like connecting wire, screwing in wires, opening panels etc. Choose an appropriate design for the gripper based on these requirements.
    - **Actuators:** Use motors/gearboxes for the movement of the arm links and joints.
    - **Sensors:** Attach sensors to the gripper for inspecting transformers or damaged components. There should be a camera and a coil like sensor mounted on the gripper.
4. **CAD and Analysis:**
    - Use Fusion 360 or SolidWorks to design the robot.
    - Sketch the robotic arm links to determine appropriate lengths and ensure it fits within the required work envelope.
    - Conduct static structural analysis using Ansys for both the chassis and the robotic arm to identify and address potential failure points.

General Instructions:

1. Develop a chassis that supports the robot's weight, including the robotic arm and spare components.
2. Ensure the design includes compartments for electronic boxes and spare parts.
3. Design a suspension system for continuous tracks or wheels to handle stairs and uneven terrains.

Bonus:

- Show detailed torque calculations for the required motors.
- Extra credit if you get creative and choose to implement other mechanisms like for example legged robots.
- Clean modular design with design for hassle-free routing of wires to motors, wheels etc.

4.2 Electronics

Objective:

Design the complete electronics system for an autonomous electrical surveillance robot using TinkerCAD simulation. The system should be efficient, reliable, and capable of controlling the robot's movement and functions.

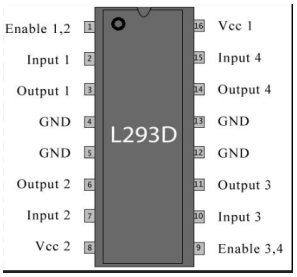
Requirements:

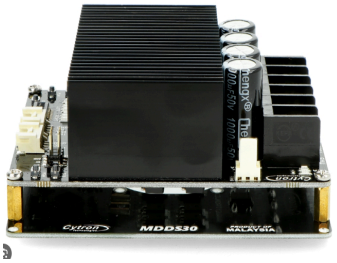
1. **TinkerCAD Simulation:**
    - Design the entire electronics system using TinkerCAD.
    - Assume the components in TinkerCAD have sufficient torque, power, and other features.
    - Use one Arduino UNO.
2. **Traversal:**
    - Select motors based on torque calculations from the mechanical design.
    - Choose a motor driver that can deliver the required power to the motors.
    - Implement a driving mechanism accordingly (ex: differential drive) to control the robot.
3. **Code:**

    Include thorough comments explaining the code.
4. **Sensing:**
    - Wireless current sensing using Hall-effect probe measures
5. **Power Calculations:**
    - Calculate the power requirements for the robot.
    - Choose and justify the type of battery (e.g., LiPo, LiFe, NiCad).
    - Calculate peak current and determine the ideal gauge of wires.
6. **PCB Design:**
    - Design an Arduino UNO shield with the required footprints and components to minimize wiring.
    - Use screw terminals where wires need to attach to the PCB.
    - Use KiCad for PCB design.
    - Make thicker tracks where needed and justify the chosen thickness.

General Instructions:

- Make and document any necessary assumptions.
- Specify the gauge of wires and the expected peak current.
- Provide relevant calculations.
- Ensure your PCB design matches the actual components.
- Note that TinkerCAD components might differ from actual components (e.g., using L293D in TinkerCAD but requiring a Cytron in reality).
- No need to do any electronics for the robotic arm (assume triggering on 1 relay picks up the object and another relay drops it).





Bonus Points:

1. Lower electronic power consumption.
2. Elsewhere specified earlier.
3. Doing electronics with minimal use of Arduino (only for serial communication). Basically try to do it without arduino but you'll need it just for serial communication.
4. Complete 3D model of your PCB with all components present.

Submission Format:

Submit your work in a Google Doc with a small report including the following table:

Component Name	Component Image	Component Justification	Component Power/Specification	Component Link

- Include all components in the table.
- Your report should include all relevant calculations.
- Include screenshots of pcb(schematic, wiring and 3d).
- Share the tinkercad file.

4.3 Automation (Perception, Planning, Control)

Objective:

Design and implement the automation system for an autonomous electrical surveillance robot using Turtlebot 3 and Gazebo simulation. The robot needs to navigate to designated dysfunctional electrical sites marked by cones and carry out surveillance tasks. The core tasks include path planning, control, and computer vision.

Task Instructions:

1. **Setup:**
    - Use Turtlebot 3 and Gazebo for simulation. Instructions for setting up Turtlebot 3 in Gazebo can be found in repo.
    - The starting position of the robot is (109, 296) in the Python OpenCV coordinate frame.
    - The coordinates of the cones are as follows: (subject to change)
        i. (159, 208)
        ii. (296, 142)
        iii. (502, 539)
        iv. (244, 640)
        v. (190, 326)
2. **Cone Detection and Status Reporting:**
    - Coloured cones are placed at the surveillance sites:
        i. Blue Cone: No error at the site.
        ii. Red Cone: Erroneous site detected.
    - Use the camera mounted on the robot to detect the color of the cones.
    - Publish the following strings to the ROS topic `task_status` based on the cone color.:
        i. Blue Cone: "No error detected"
        ii. Red Cone: "Error detected at site"
    - After the message for the final cone is published, output: "Surveillance task completed"
    - Bonus: Define and use custom messages to send site status and location.
3. **Path Planning:**
    - Develop a path planner using a sampling-based algorithm (e.g., RRT, RRT*).
    - Alternatively, create your own map for graph-based algorithms such as A*.
    - Plan a path from the robot's starting position to each marked location.
    - Ensure the robot avoids obstacles and does not collide with the cones.
4. **Controller:**
    - Implement a PID controller to ensure the robot follows the planned path accurately. (You are free to explore other controllers too).
    - Add control logic if necessary to prevent collisions with cones.
5. **Obstacle Detection:**
    - Use the provided function `isValidPoint(parentX, parentY, childX, childY)` from `obstacle_detection.py` to check the validity of points in your path.
    - Utilize the provided `visualize_path` function to visualize the planned path.
    - Once you have planned your path, use the `point_transform` function to

transform the points from the python OpenCV coordinate frame to the gazebo simulation coordinate frame. Make sure to use these transformed coordinates for your controller.

6. **Colour Detection:**
   - Use OpenCV to detect the colors of the cones from the camera feed.
   - Implement this in the node named `colour_detector`.

7. **ROS Nodes and Topics:**
   - The following nodes exist in the src folder of the ROS package `hackathon_automation`:
     i. `path_planner.py` (Node name: `path_planner`): For path planning.
     ii. `controller.py` (Node name: `tb3_controller`): For controlling the robot along the planned path.
     iii. `colour_detection.py` (Node name: `colour_detector`): For detecting cone colors.
   - The planner node should publish the planned path to a topic (e.g., `planned_path`).
   - The controller node should subscribe to this topic to receive and follow the path.
   - That is, in detail:

Name of **Node** (not file)	Subscribing to	Publishing to
path_planner		/planned_path
tb3_controller	/planned_path, /odom	/cmd_vel
colour_detector	/camera/image_raw	/task_status

General Instructions:

- Adjust the coordinates of the nodes in the planned path using the point_transform function for each (X, Y) coordinate.
- Begin planning from the initial position (109, 296).
- Do not account for the size of the robot while planning; obstacles have already been adjusted accordingly.
- Ensure to document any assumptions you make during the task.
- Provide all relevant calculations and explanations in your code comments.

Bonus Points:

- Define and send custom messages with site status and location.
- Implement efficient and optimal path planning and control to reduce power consumption.

4.3.1 Environment Setup

To set up the simulation environment, the following packages are required:

- Turtlebot 3
  Follow setup instructions here
- Clone the hackathon repo in your catkin workspace source folder and then run colcon build.

The launch file needed to launch the world as below is provided in the ROS package named `hackathon_automation`. Run the following block of code to launch the world with the turtlebot along with the three nodes.

```
ros2 launch hackathon_automation robo.launch.py
```

To launch the respective nodes you will need to add appropriate commands in the launch file. You should be familiar with this once you go through the tutorials.

5 Final Points

All this might seem a little intimidating if you haven't worked on a big project before. The key is to break it up into manageable chunks. We recommend starting off by building a small publisher that publishes to `/cmd_vel` to move the robot. Once you are comfortable with this and can see the robot move properly in the simulator, you can proceed to work on a path planner. After the planner is ready, you can separate everything into individual ROS Nodes.

Another essential skill you should develop is "The Art of Googling" (not even kidding). Platforms such as Stack Overflow and ROS Wiki make solving errors and debugging much easier. Therefore, we highly encourage you to google your errors and queries first before asking any of the seniors.

The steps listed below are just hints on how to proceed. Feel free to follow any method you prefer (make changes in the launch file accordingly):

1. **Start Simple:**
    - Build a small publisher to `/cmd_vel` to move the robot.
    - Ensure the robot moves properly in the simulator.
2. **Develop the Path Planner:**
    - Once comfortable with basic movement, work on the path planner.
    - Use sampling-based algorithms like RRT or RRT*, or create your own map for graph-based algorithms like A*.
3. **Separate into ROS Nodes:**
    - After your path planner is ready, separate your code into individual ROS Nodes.
    - Implement nodes for path planning, control, and colour detection.
4. **Debugging and Problem Solving:**
    - Practice using online resources like Stack Overflow and ROS Wiki to debug and solve errors.
    - Google your issues first before seeking help from seniors.

Note:

Although all three tasks (mechanical design, electronics interfacing, and path planning) are related, they are independent of each other for the purposes of this hackathon. The electronics interfacing or the path planner does not depend on the mechanical design in this hackathon (although in a real-world system, all three would be interconnected).

Finally, this hackathon is meant for you to learn and have fun! We hope that you have a great time tackling the challenges and enjoy the process