Computer Science 370 Operating Systems
Fall Semester 2023

December 6th

Raspberry Pi Rover

Submitted by:
Jack Adams jadams37@colostate.edu
Josh Circenis jecircen@colostate.edu
Daemon Kerrigan dk2019@colostate.edu

Professor: Shrideep Pallickara

CONTENTS

INTRODUCTION

For this project, we were challenged to solve a problem by leveraging the capabilities of a Raspberry Pi in conjunction with other components such as sensors and other computers. Additionally, we wanted to solve a problem that would be widely applicable and useful to a large audience of potential users. To achieve this, we decided to work on a very broad problem and develop a solution that is flexible for specific needs. So, the problem that we set out to solve was remote data collection and monitoring. This problem is increasingly more prevalent in society today as data is used to make policies, research hypotheses, discover problems, and predict patterns and trends for the future. Because of this, a suitable solution to this problem could have a great impact in many ways by aiding in creating a more efficient, effective, and streamlined tool for large-scale data collection.

PROBLEM CHARACTERIZATION

To create an effective solution to this problem, we first have to understand the issue and the users' needs. Because data can be collected in many different ways and take many forms the solution must take this into account to be useful for a user. For example, a researcher collecting ambient noise data will have different sensors and

data collection methods than a researcher who is monitoring background radiation levels in a certain area, but the solution should be useful to both researchers. This exposes one of the main requirements that our solution must meet in order to be an effective solution, flexibility. The solution must be flexible enough to be adapted to multiple different data types and collection methods. If this goal is not met then our solution will have limited applications and will not be as useful to a potential user. Additionally, the solution should be easy to use both in terms of operation and setup for the user. Since flexibility is a core goal it must be easy for the user to adapt the solution to their needs in the setup and after setup it should be easy and intuitive to interface with and operate in the required manner. If this goal is not met, then while the system would be versatile and able to be used in many ways users who do not have technical experience would not be able to use the solution for their needs, so it would have limited accessibility and would not be practical for many potential users. Similarly, the solution should be cost-effective in order to be maximally accessible for any potential users. If these goals are adequately met, then the proposed solution will be adaptable to different needs, easy for anyone to use, and accessible both in terms of required skills and cost. This will maximize both the number of potential users and the usefulness of the solution to those users.

PROPOSED SOLUTION

Initially, we were interested in solving this problem by utilizing communication between two Raspberry Pis. The initial project framework relied on the two Pis communicating over separate networks. One Pi would act as a user interface that would retrieve sensor data from the other Pi and present it in a useful manner for the user. This system, depending on the configuration, could be adapted in many different ways to suit the user's needs in remote monitoring or data collection. In researching a solution to this problem we were able to find many solutions, however, many of these were not feasible within the constraints. This is because many of the solutions found rely on external services or modifying network configuration. This presents several issues, such as exposing network vulnerability, limiting accessibility, and increasing complexity. Because of these issues, the solution would not have adequately met the previously stated goals of flexibility and ease of use for the user. Additionally, the system is not entirely user-friendly as it would require a good understanding of Raspberry Pis and programming in order to set up and operate. So, we decided that it was not practical to pursue the idea further and decided to pivot to something else.

Because of the previously stated issues with cross-network communication, we instead looked into local communication. Making this switch allowed for the flexibility and ease of use that we wanted, but limiting communication to the same network while maintaining the same structure eliminated the core purpose of the project. This is

because the solution would no longer be necessary as it requires proximity to the data collection and monitoring thereby eliminating usefulness. This led us to completely rework the project in terms of the purpose, structure, and components used. To do this, we first looked at the resources available to us in order to further develop an idea that would be feasible to implement. After looking at the options available to us through Josh's history working with electronics and robotics we decided to create a mobile system that would be able to be utilized over a larger area even under the constraints of local network communication. This can be achieved by mounting the Raspberry Pi and other necessary components on a rover which is controlled by a device on the network. To fit our goal of ease of use we decided that most of the processing should be done on the Pi which would relay the results to the user in an intuitive and understanding way. To accomplish this, the Pi is hosting a locally accessible webpage. This webpage provides the interface for the user to the rover and serves the purpose of allowing the user to send commands and receive results. This is ideal because for less technical users the interface is accessible from any phone or computer and for more technical users it would be easy to write software to interface with the webserver. To meet our goal of flexibility the rover utilizes a serial connection to an Arduino board that interfaces with the hardware. This solution is a balance between flexibility, cost, and technical requirement because the Arduino is inexpensive, easily programmed to suit different needs, and different boards can support the required functionality. However, this does mean that the user may not have the required technical ability but the Arduino IDE is readily available and there are many resources on how to use and program Arduino which is why this solution was chosen. Additionally, with this solution, a more technical

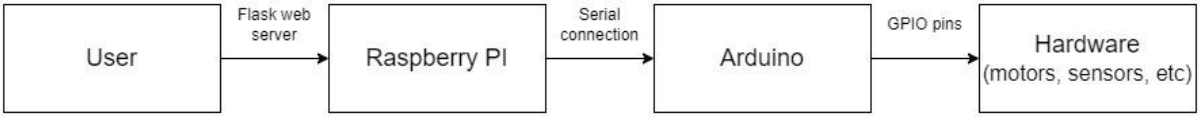
user could utilize the other serial ports on the Raspberry Pi to interface with multiple devices at once and increase the ability of the rover to accomplish a wide variety of tasks.

IMPLEMENTATION STRATEGY

In order to build an interface to interact with the rover we first need the ability to send commands to the Raspberry Pi. To achieve this, we began looking for a good way to host a locally accessible web page from the Pi. There are many ways to do this, but we decided that we would use a web application framework written in Python called Flask. We chose to use Flask because it is, lightweight, simple to use, good for a Raspberry Pi, and flexible in its applications. Using the Pi we were able to create a Python file that imported the Flask framework and hosted the server. Then using a front-facing HTML page we enabled the server to run Python functions upon form submission. At this stage of implementation, the page just had a text box with a submission button. The button would evoke a request method POST and would then execute the code that would be added underneath.

Now that it was possible to wirelessly send commands to the Raspberry Pi, we could then implement the supporting code for those commands. Since we were planning to use an Arduino to interface with the hardware, the first step was to connect the Raspberry Pi to the Arduino. This was done by utilizing a serial connection between the Arduino and Raspberry Pi's USB ports. The Arduino uses a UART protocol (universal asynchronous receiver/transmitter) to communicate so the Raspberry Pi utilized Python's pyserial library which allows it to establish a connection to the Arduino

for both read and write serial operation. This serial communication is then received by the Arduino and is used to perform the accompanying task.



The diagram above depicts the structure of this project. At this stage in the implementation, a device can send and receive messages to and from the Raspberry Pi through the web server and the Raspberry Pi can communicate with the Arduino via the serial connection. Then the Arduino can interpret the commands sent from the Raspberry Pi and perform the necessary tasks. This structure is seen in many different places and utilizes an abstracted structure where at the highest level the user is sending human commands and at the lowest level is directly controlling the physical hardware. At this point, the desired functionality can be implemented on the Arduino. To make the system as useful as possible for a variety of applications, the robot will perform all of the necessary tasks without interaction from the user aside from the aforementioned commands.

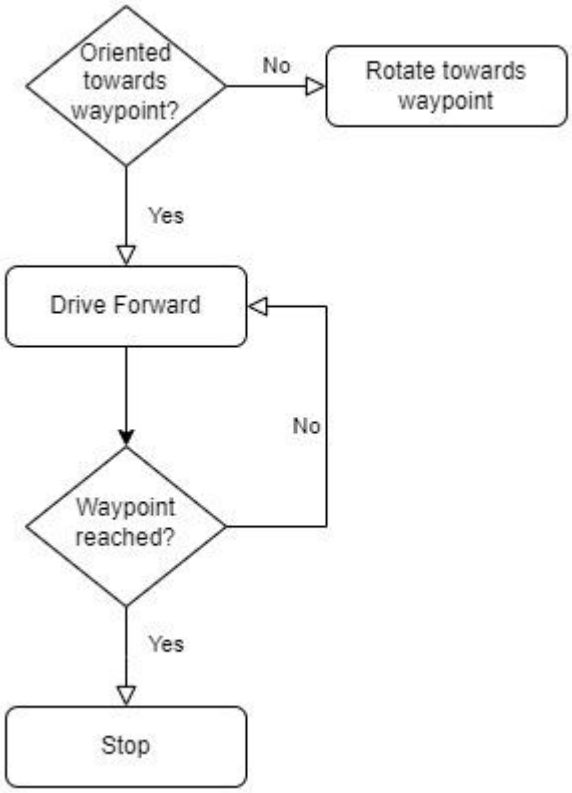
The first step is to implement autonomous movement. This will be done using a waypoint system where the user can send geographic coordinates to the rover and it will autonomously navigate to the point. For the rover to do this it first has to know where it is located and where it needs to go. This will be done using a NEO-6M gps module which is compatible with the Arduino using the “TinyGPS++” and “SoftwareSerial.h”. The gps module can determine the coordinates of the rover, the

number of satellites in range, the speed, and the course. This is enough to accurately maneuver to the given coordinate autonomously, however, at slow speeds the speed and course function are not accurate. This means that the rover will need additional sensors to complete the task. To get the heading, the rover is equipped with a magnetometer using the “Wire.h” and “HMC5883L.h” Arduino libraries. A magnetometer is a digital compass and after calibration, it will serve as an accurate way for the rover to determine its orientation. Now that the rover is capable of obtaining the necessary information to navigate autonomously we can begin to implement the autonomous navigation. The first step is to set up the necessary motion functions for the rover. The motor controller utilized in this rover is the “L298N” dual channel h-bridge type controller to control two groups of two motors arranged in a differential drive. This means that the direction of each motor is controlled by two digital outputs and the speed is controlled by a pwm output. The rover is configured such that two of the four motors are grouped in pairs giving us differential (tank) steering. The next feature to implement is a “rotate to heading” function. For the rover to navigate autonomously it will need the ability to autonomously rotate to a desired heading. This was done using only the proportional term of a PID loop with the magnetometer orientation as an input and the pwm levels as an output. To efficiently rotate to the heading we first must determine the optimal direction to turn, clockwise or counterclockwise, to reach the designated angle faster using the equations “(current angle - desired angle + 360) % 360” and “(desired angle - current angle + 360) % 360” where “current angle” is the orientation of the rover in regards to magnetic north and “desired angle” is the desired heading of the rover. Using the equations to determine the direction to turn we now know the direction to set the

motors, but we still don't know the speed. To get the appropriate speed we must take into account how far (in degrees) we are from our target heading. This allows the rover to set its speed to not overshoot the target but instead to speed up and slow down proportional to the distance from the target in a feedback loop. This is done with the following equation for counterclockwise
"| | 360 - desired angle | - | 360 - current angle| * 255" and "| desired angle - current angle| * 255" for clockwise where 255 represents the full speed of the motors. Since the rover can now move and turn accurately we need to be able to determine where to go and how to do this. The first value that we must compute is the bearing (angle) of the line formed by the coordinates sent by the user and the coordinates of the rover. This can be done using the bearing equation derived from the Haversine formula (Upadhyay, A). Then the Haversine formula can be used to determine the distance between the two points. With this information, it is possible to create the control loop for autonomously navigating to the target coordinates.

The control loop utilized for the rover is as follows, first, the rover gets the bearing to the defined point and then orients towards the point, it then drives forward until it reaches within a specified error of the point or goes off course. We can determine if the rover is off course by getting the heading from the magnetometer. If this value is outside of a certain range from the target then again the rover will calculate the heading reorient and drive forward until the goal is reached or another course correction must be made. In this manner, the rover autonomously navigates to the desired location correcting its course when necessary. The flow chart below shows the navigation

control algorithm where “Orient towards waypoint?” and “Waypoint reached?” are checked continuously over the period of navigation.



Now that the rover can be told where to go and navigate to the point autonomously, the autonomous base is complete. At this point, the user is able to create their own custom operations or data collection methods and then have the rover complete these tasks by assigning a corresponding command. Once the Arduino sends a message back to the Raspberry Pi over serial, we use the serial library’s readline function, as well as some other functions to decode the response, to get what the Arduino sent back. The code then inserts the response at the front of a list for storing commands for the HTML code to process. At this point, we added a response section to

the webpage so that we could display the Arduino's response. Using the list of responses in the Python code, we add a for loop in the web page code that displays each received message on a new line.

CONCLUSIONS

For this project, we set out to create a user-friendly and cost-effective solution for remote data collection and monitoring that is flexible enough to be easily adapted to specific needs. Our solution to this problem was an autonomous rover that a user can send commands to through a web server to complete tasks. Furthermore, this solution is very modular because its core structure relies on a series of abstractions which simplifies the structure and makes the project more flexible. Additionally, because all of the components are easily accessible this solution is cost effective and user friendly. However, this solution could still be improved upon. For example, if a more accurate GPS module and magnetometer were used then the navigation could be improved through the accuracy of the sensors. Additionally, the navigation control loop could be improved by adding steering while moving to the control loop and only stopping to turn when further off course, thereby improving efficiency. But ultimately these reduce flexibility while our current solution has the ability to adapt to multiple drive bases, motors, and motor controllers. However, one aspect that could be improved to increase flexibility is the webpage design. Multiple different methods of control could be added to the web page to increase the number of ways that the user can control the rover and the flexibility with which the system interacts with the user interface. However, in all

other regards, our solution has stayed true to our original goals and fits the problem design.

BIBLIOGRAPHY

Flask. Welcome to Flask - Flask Documentation (3.0.x). (n.d.). https://flask.palletsprojects.com/en/3.0.x/

Pyserial. PyPI. (n.d.). https://pypi.org/project/pyserial/

Upadhyay, A. (2022, November 18). *Formula to find bearing or heading angle between two points: Latitude longitude*. -. https://www.igismap.com/formula-to-find-bearing-or-heading-angle-between-two-points-latitude-longitude/