Pluto Rover - Autonomous Navigation

Pallav Hingu

Contents

Chapter 1

Analysis

1.1 Description of the Project

Space Exploration has advanced massively in recent years. Large companies such as NASA and SpaceX are taking large steps forward to explore and colonize new planets such as Mars. We know a lot about the planets in our solar system: We have pictures, samples and critical data about the atmospheres of them. This has opened a gateway allowing us access to further resources from the planets in our solar system. But, there is one "planet" that we have yet to explore in detail; a "planet" where no man-made object has never landed: Pluto. Having been recently described on Google as: *"Our favourite dwarf planet since 2006"*, we have yet to discover its surface and internal composition. To continue our research of the planets in our solar system, we will need to **send robotic rover on a research and exploration mission to Pluto.**

As Pluto is extremely far away from the Earth, there are many obstacles to sending a rover to Pluto. Whilst the most obvious is transportation, NASA have already proved that they are capable of doing this using their "New Horizons" interplanetary space probe launched as part of the "New Frontiers" program. But, after getting to Pluto, there are many challenges that a rover can face. Firstly, due to the large distance between Earth and Pluto, there is a considerable delay in receiving signals, and there is also a lot of data loss. This means that the rover cannot easily be controlled remotely as there would be a large delay in sending instructions as well as receiving camera/sensor feed, which means that the camera may not represent the actual situation of the rover when the feed is received on Earth. As this poses a large threat to the condition of the rover, the best approach to address the safety of the robot is to install an autonomous navigation software, which avoids all obstacles in it's path and uses other available sensors on the rover to get more information on the surrounds.

The information received using these sensors will be stored on the rover as well as sent to the nearest orbiting satellite, which would have been used to transport the rover as well. This satellite will then also store the data received from the rover's sensors and then will broadcast these results to Earth, allowing it to be analysed.

I will design a software with a friendly user interface (which will be on the companion app) that presents live information from the rover on a remote device, which can be on a satellite to monitor the rover and receive crucial data. The rover is on Pluto to detect the presence and location of metals on the surface. My software will utilize IoT protocols to present the output of the metal

detector. It will also use a built-in IR camera to perform autonomous driving. A smart algorithm will also be used to generate the relative location of the rover on Pluto's surface according to its set landing location and landing orientation. This will allow the rover's position to be estimated, as GPS cannot be utilised on Pluto.

The aim of this mission is to detect if there are metals near the surface of Pluto, if yes, then to identify and record where.

1.2 Computational Methods

I think that this problem lends itself well to the computational methods such as abstraction and decomposition for a variety of reasons. The solution will be a software that controls the Pluto rover on a hardware level, allowing it full autonomous control as well as manual, using the rover's cameras and sensors. This will simply need to run on a computer as that will be the medium on which the images from the camera will be processes and the next course of action for the rover will be determined. There is no alternative solution for this problem that will not require a computer, due to the context of the problem, and the environment in which the solution will need to function (the harsh climate of Pluto).

1.2.1 Methods Used

Some of the computational methods used for the solution are:

- **Problem Decomposition**
  This large problem can be decomposed into many smaller steps. These steps can be thought to be the following:

  1. Use the rover's camera system to get some RAW data about the surroundings
  2. Add threshold to data to limit the amount of data being passed through the program
  3. Plot an image representation of the RAW data
  4. Use image processing to detect the obstacles in the image
  5. Use a smart algorithm to determine the action needed to avoid obstacles
  6. Send instructions to a driver board (allowing hardware control to motors)
  7. Use an algorithm on the driver board to control the motors
  8. Use a feedback loop to get a confirmed movement of the rover
  9. Calculate vector location of rover using its movement
  10. Get data from the metal detector
  11. Write vector location and metal detector data to local file
  12. Send data to server for companion app

  After these steps are completed, the problem can be said to be solved. The steps will allow the rover to react to obstacles in its surroundings, allowing for autonomous navigation. Also, the steps allow the rover to utilise a metal detector and send its data to a server for analysis, meaning that we can get more information about the surroundings of the rover and the surface of Pluto. The vector location calculation will also allow for the rover to be located using no other hardware components, using only the velocity and angular velocity of the rover.

- **Problem Recognition**
  The overall and apparant problem is autonomous navigation through an extreme terrain. But, the underlying and primary problem is the detection of obstacles and their location. If the obstacles can be located, this information can be passed through a set algorithm that will allow a motor response.

- **Thinking Procedurally - Divide and Conquer**
  All the decomposed steps above allow for the problem to be divided into smaller problems, which are easier to solve than the large overarching problem. Creating functions/modules to solve the smaller problems, and them combining them all a larger modular program is an example of divide and conquer as the large program had been divided onto smaller problems, that each formed modules that could easily be combined.

- **Abstraction**
  Abstraction plays a large role in ensuring that the program responds to input data immediately. The incoming data stream from the camera sensor contains a lot of information, most of which is irrelevant for this application. So, it is vital to utilise abstraction and extract useful information only from the camera's input data. In this case, we would add a threshold so that the data only represents obstacles a certain distance from the camera. This allows us to reduce the amount of information being fed into the program, decreasing its response time and increasing its speed. The obstacles that are detected by the camera but are further away from the camera are removed as they do not need to be considered yet, they do not pose a risk of collisions yet.

- **Logic Modelling**
  Logic modelling will allow me to illustrate how the program must work to fulfil the system requirements. It will also be helpful to depict the actual program's activities compared to its intended needs, allowing me to measure how successful the program is at fulfilling it's function and solving the problem. As this will help with the application evaluation, it will be useful when improving the program as it ensures that all the requirements are fulfilled well.

1.3 Stakeholders

Not done yet.

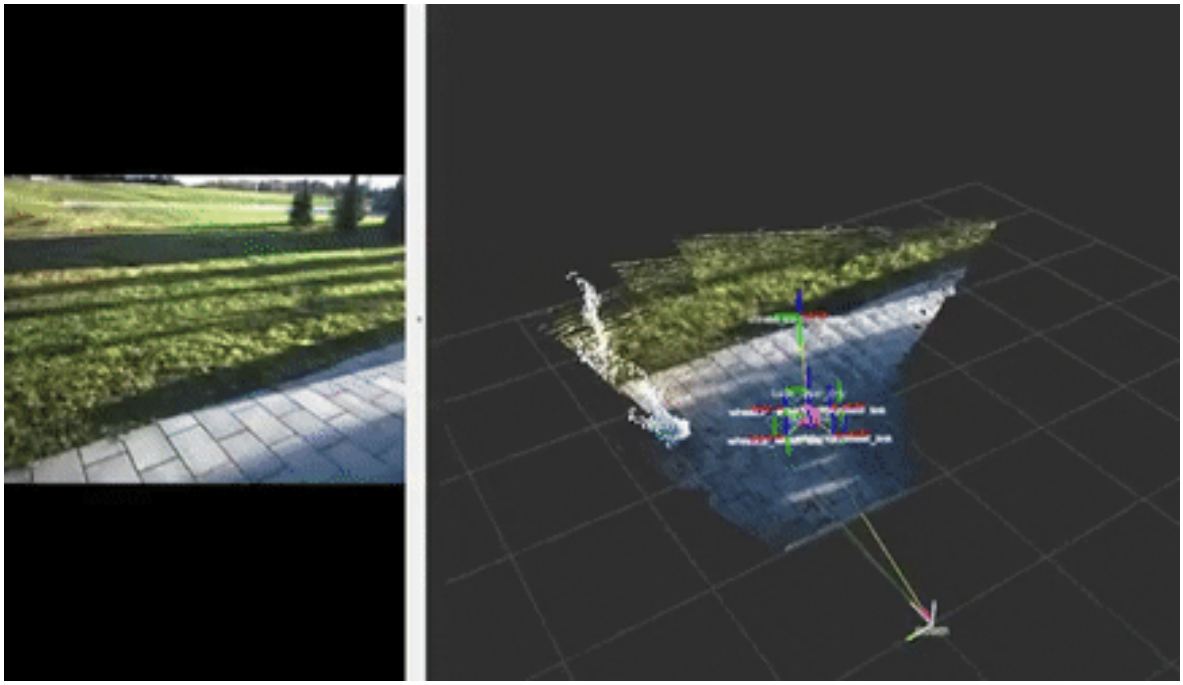
1.4 Research

1.4.1 Exisiting Solutions

As this is not a commonly commercially developed program, there are very few reviewable examples available online. It is also very specific to the context, which may be a reason why the program is uncommon. But, if we widen the scope and look for software that are not specifically designed for this application but can be used in this context, we can find more programs online.

Examples:

- **RTAB-Map with ROS**



Overview

RTAB-Map (Real-Time Appearance Based Mapping) is an open-source application which uses different SLAM (Simultaneous localisation and mapping) approaches to generate a 3D dot-cloud of the surroundings of a sensor. It can be installed and used as a stand-alone application to generate 3D dot-clouds based on sensor inputs, or can be integrated into ROS (Robot Operating System) to allow the application to control hardware according to the 3D dot-cloud generated by a camera/sensor on a robot. It produces a 3d Map of the surrounds and uses localisation to detect the robot's location in relation to the 3d generated map. it then uses this to detect obstacles and find the most efficient way around the map to a given goal location.

ROS, which is essential in this application, is a robotics middleware and allows the RTAB-Map here to publish real-time messages to the appropriate nodes. In ROS, a node is a process that

performs computation. Nodes are combined together in a graph and can communicate with each other to perform certain tasks. RTAB-Map, when integrated well with ROS, publishes messages to many external nodes, which communicate with hardware drivers in the system to move the robot according to RTAB-Map's instructions. As RTAB-Map can get a large image of the surroundings, it uses a smart algorithm to conduct path finding, meaning that it calculates the most efficient route to reach a destination while avoiding obstacles.

As this is a complicated process, it requires a lot of advanced hardware and a high amount of processing power. ROS is also extremely difficult to use, as well as install correctly. ROS is used for many complex robots, like the Robonaut 2 aboard the International Space Station.

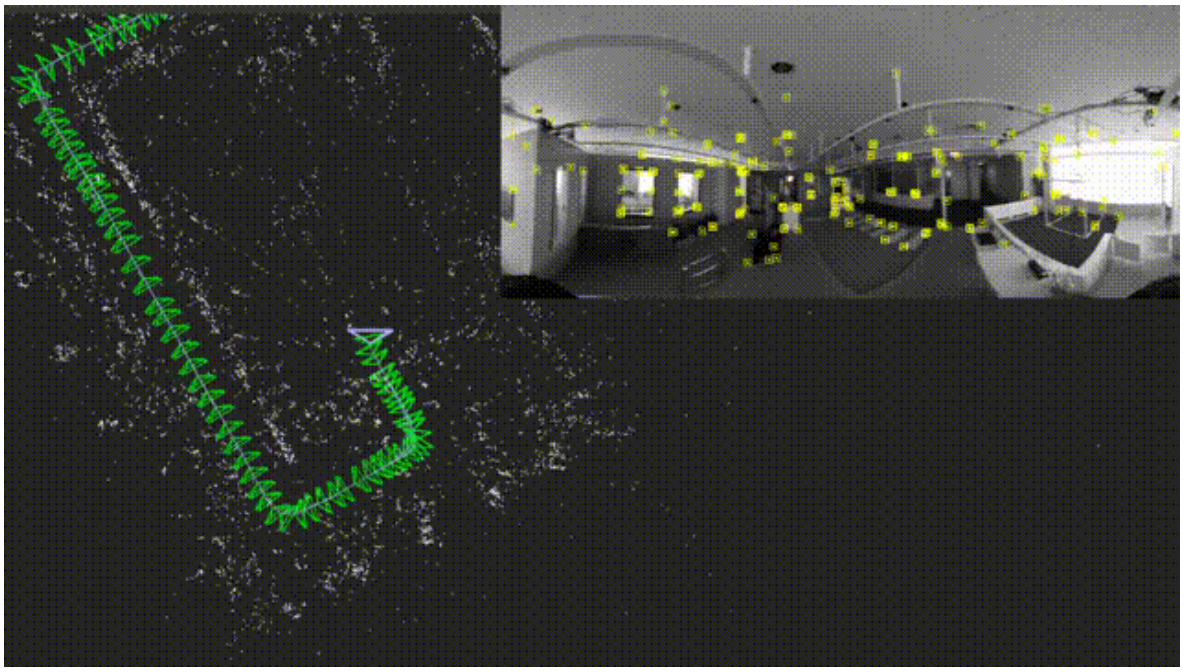
The Graphical User Interface(GUI) that it uses displays lots of information, such as the 3d Dot-map created of the surroundings in real-time, the raw images that are being captured, and the calculation of odometry (the relative location of the vehicle). This is a simple but effective and advanced user interface that provides a lot of data that can be useful for environmental analysis, but is not hard to use. As this application is not specific to a problem, the GUI can be adjusted to display less or more information, or different information by tweaking the settings. This allows the application to be adapted to many different situations, making it ideal even for an autonomous Pluto Rover.

Parts that I can apply to my solution

From this application, there are many things that I can use in my application:

- *Display Odometry* - the application displays the odometry (relative location) of the vehicle on the GUI. This will allow the relative location of the rover to be displayed on the GUI so that the Rover can be located.
- *Hardware Control* - the software can control hardware directly by converting the output into electrical signals for the components. This will allow the Rover's motors to be controlled by my application directly.
- *Navigation* - this application can determine movement to navigate around a map with a set objective. This will be useful to move the Rover around the surface of Pluto with an objective to detect metal.
- *GUI settings* - the use of GUI settings allows the displayed infromation on the GUI to be adjusted according to user preference or need. This can also be helpful in low resourse scenarios as displaying less information means that the program runs smoother and faster. This could be helpful on the Rover as there are limited computational resoursesand so the adjustment of the GUI to the scenario may be helpful in improving performance.

- **NVIDIA DRIVE**



1.4.2 Primary Research

Questions for Interview

Interview

1.5 Features of the Proposed Solution (System Goals)

1.6 Software and Hardware Requirements

1.7 Success Criteria

1.8 Limitations