Clarkson University
ECE Department
10 Clarkson Ave, Potsdam, NY
EE 416/464 Computer Engineering Senior Design

PACMAN

Path **A**ssembling **C**artographer, **M**apping under **A**utonomous **N**avigation

Final Analysis and Evaluation

Very Good Robotics

Max Cutugno, Avery Oefinger, Adam Romlein, David Russell

Fall 2019

Table of Contents

1. Introduction

The PACMAN is an autonomous robot for surveying and path mapping. Described within this document is the evaluation and design that our team, *Very Good Robotics*, produced. This evaluation will give a detailed description of the specifications met in our initial design document, as well as those failed. Any additional capabilities reached will be mentioned, and why they were implemented. All system testing and results are outlined at the end of the document. All source code and hardware manuals for this project are attached in the appendix.

2. Evaluation and Analysis of PACMAN

PACMAN's various requirements were outlined in our initial design system requirements and specifications (SRS) document. In the following sections we outline which areas PACMAN succeeded in, and which areas require more time to be considered a success. Green indicates a completed requirement, orange indicates a partially completed requirement, and red indicates a missed requirement.

2.1. Functional Requirements

This section provides how the system must operate in order to be considered a successful design, i.e. "What the system shall do.

2.1.1. Shall have both an autonomous and manual operation mode.

2.1.1.1. Shall detect and stay on paths while in autonomous mode.

2.1.2. Shall cause no damage to either persons or personal property while under operation.

2.1.3. Shall cease motion immediately if the operator requests it.

2.1.4. Shall have a watchdog timer onboard communicating with a base station computer, timing out after 3 seconds and ceasing all motion.

2.1.5. Shall acquire real-world coordinate location of paths traversed.

In the area of functional requirements, our system performed quite well. All requirements were met, except for the one sub-requirement 2.1.1.1, our system does not always stay on the paths in autonomous mode. It can generally travel a brief distance (20-100ft) on a straight path before losing its bearings, but it is not consistent enough to be called a full success.

2.2. Performance Requirements

This section defines how the system will physically and computationally perform to be considered a successful design.

2.2.1. Shall autonomously traverse a path at a minimum of 1 meter per second.

2.2.2. Shall contain entire 3-D physical space occupied by itself within the path.

2.2.3. Shall know its relative location within 10 centimeters.

2.2.4. Shall have the ability to sense objects in motion near operational area at a minimal rate of 3 Hz and cease motion if any are detected within 2 meters.

2.2.5. Shall operate under battery power for at minimum of 3 hours.

2.2.6. Shall inform operator when at low battery.

2.2.7. Shall know its absolute location in real-world coordinates within an accuracy of 5 meters.

2.2.8. Shall transmit to base-station the real-world coordinates of all sidewalk paths it traverses, within an accuracy of 5 meters.

A few performance requirements were not achieved within our timeframe. The 1 m/s speed requirement was not met, the speed was throttled to 0.4 m/s due to safety concerns. Remaining completely in the path was discussed before, and the 20-100ft consistency applies here as well. No depth detection was implemented in the course of this semester.

2.3. Elective Requirements

This section defines what the system could achieve if all functional and performance requirements are satisfied prior.

2.3.1. More noticable audible or visual presence.

2.3.2. Creating a 2D image of the paths and area it has traversed.

2.3.3. Interfacing with a GPS beacon to acquire sub-centimeter accuracy.

2.3.4. Measure the width of the sidewalk traveled.

None of these elective requirements were fully completed. PACMAN's visual presence was made obvious, but additional flashing lights or speakers could have been added. A basic implementation of overlaying images in the space over which PACMAN had traveled was tried, but the final product wasn't finished. Multiple attempts were made to interface with the on-campus GPS beacon to obtain corrections, but none of these were fruitful with an output. The width of the sidewalk is already obtained via our segmentation and calibration, but is not output to the user in any meaningful way. Adding this in would be an easy addition.

2.4. Attributes

2.4.1. Reliability

PACMAN had its reliability issues early on, with random shut offs and controller disconnects, but as the system has progressed it has turned into a very reliable, well-constructed system. All runscripts start at boot, and the controller can be paired with no SSH connection first required. The random shut offs were fixed with a dedicated voltage converter for the Jetson Xavier, which was picky with the 1V ripple wave 12V supply, so was upgrade to a more stable, 19V supply

2.4.2. Maintainability

PACMAN is easy to maintain with a system made from wood, and can be unscrewed and any component can be accessed with ease.

2.4.3. Adaptability

Since PACMAN is easy to access and modular, components can be swapped out, upgraded, or new components added to adapt to any situation.

2.4.4. Safety (user must be present for autonomous mode)

PACMAN is equipped with numerous emergency stops, both physical and remote. A tier of watchdog timers are implemented to shut off power if any crucial component stops communicating. However, the operator should always be present under autonomous operation.

2.4.5. Portability

Since PACMAN is on a wheelchair base, it can access any place with handicap accessibility. It can also be lifted by 2 people.

2.4.6. Presentability

The wiring of PACMAN is kept clean and presentable, with all components contained within the enclosure. The weatherproof components are kept on top, and vulnerable components are contained within.

2.5. Unique Innovations

Here we describe the unique innovations in hardware and software that we made to achieve our design goals and how they were implemented.

2.5.1. Navigation

Our approach to path planning, and navigation in general, was unique because we built our navigation stack from the ground up. There are 6 main steps necessary for full autonomy:

1. Motor Control via Non-User Input

   We accomplished this via our ability to enter velocities in ROS's command velocity input, which is agnostic to the specific device the velocity comes from. Whether from the controller or autonomously generated, the same output is generated.

2. Odometry

   Our odometry was generated from 3 different sources: motor encoders, Piksi IMU, and the Ouster IMU, and all fused together via an Extended Kalman Filter to generate a very high-resolution state estimate.

3. Locating the Path

   We are able to determine the path quite accurately with the Semantic Segmentation algorithm used from MIT. With some modification, this is returned as a probability image, where each pixel intensity represents the confidence that that pixel is path.

4. Calibrated Camera

   We assume that everything is in the same plane in image space, and then select the floor plane from a top-down view. This is all done with an image taken on some marked, real-world space where unique correspondences can be identified.

5. Where to Place Goal Points Within Real-World Path

This is an area that could use the most improvement. Our current method selects a source point that is in the middle of the image at the bottom, and selects a sink based on the furthest point away from the robot. A path is generated between these two points using Dijkstra's algorithm, using the probability that each point is a path as the cost map.

6. Navigate to Goal Points

   Our navigation to goal points required some fine tuning to get an accurate turning system. Making a non-linear angular velocity to reach the target angle was key to achieving smooth operation. We took the square root of the angle needed to go, which helped speed up minor angles and prevented huge speed spikes for large angles.

2.5.2. Hardware

Our hardware configuration was well-suited for dealing with the high-amp loads that our system must endure, due to the powerful motors. A novel implementation was the use of two high current relays and switch to have a push-button system that can be turned on via a press and shut off via an E-stop. If the E-stop is disengaged, however, the system does not immediately resume running, but waits for another button press. All components were rated properly, and have functioned consistently well for all operation since testing. This system also places the load on the high-current relays, which are the most resilient components of the system

2.5.3. Simulation

To test our navigation stack, we had a very realistic simulation that represented our system. This enabled us to work on the system in parallel, and also to test our navigation in a completely safe environment, with no threat of physical damage. Furthermore, along with the development for simulation, a CAD model was created for several purposes. The first and most apparent use of this model was to visualize the actual geometry and texture of pacman in a simulated environment (through RVIS or Gazebo). The second use to reference measured distances between the assembled components, as the model was designed with accuracy of approximately ⅛ of an inch. With these measurements, a frame hierarchy (hierarchy of odometry transforms) for all the sensors was generated on top of the CAD model. This allowed odometry fusion to be relatively simple in comparison to creating transforms manually.

2.6. Weaknesses

PACMAN is, or course, not a perfect system. It has a few key areas that could be improved with future work. Firstly, object detection and avoidance is weak, as segmentation/path planning does not accurately generate paths around objects seen by the camera. Therefore, the LiDAR sensor would be used for this circumstance. LiDAR is captured during PACMAN's operation, however LiDAR data is never used for object detection. This is because the object detection stack was never implemented due to the semester time constraint. Another feature that wasn't implemented due to this constraint was building 3D maps from LiDAR data. The map would encapsulate the geometry of the path's surroundings that PACMAN followed during operation by merging LiDAR frames.

Another big weakness of PACMAN is the path prediction latency (2-3 seconds). This relatively long latency comes from the segmentation processing. A direct consequence of this issue was formulating a path planning algorithm that accounts for latency. The averaging method that was used works to some degree; following the farthest point PACMAN sees as a viable location to travel to works while following the farthest point unless a right turn can be made does not work. Therefore, the path planning algorithm must have some improvements to make the system robust. Lastly, the segmentation algorithm used does not recognize viable paths on wet, dark, or snowy areas. Because of this, path planning does not always generate an optimal path (since it tries to avoid these regions).

3. Test Results

Here we present the testing undergone for PACMAN and its systems. Both individual tests and overall tests are included and the way they helped improve the system.

3.1. Hardware Testing

While wiring circuitry and installing various sensors onto packman there were various tests conducted. These tests included wire continuity testing, voltage testing, current testing, and module level tests. While making each wire and constructing the electrical structure continuity testing was critical to making sure the circuitry was to spec with our design. Continuity testing also prevented any unforeseen shorting issues. Module-level testing for electrical components was done to make sure they would behave as established in their spec sheets. We made sure the relay switches trigger at 12 v, the 3 DC-DC voltage converters had the proper voltage differential and that the negative line was ground (0 v).

Once the electrical structure was assembled we used a voltmeter to test the voltages of all the power rails. Once the voltages were as expected all the sensors and higher-level components were plugged into their respective power rails. Once each component turned on we were able to use drivers to pull data off each sensor or move the motors, etc. If we were able to pull correct data off the sensors or use the motor driver to move accordingly the module testing was considered a success.

There was a problem we faced where the DC-DC converters we had could not source enough components to all the components who were being powered. This was identified by doing current testing and was fixed by replacing the DC-DC converter with a higher amperage output converters.

For menial components like the ethernet or USB hub, those were assumed to be working and when the system behaved like expected those components were marked as good.

3.2. Performance Testing

For performance, a lot of testing was performed on the navigation stack (all parts used to move autonomously) were working as expected. The navigation stack consists of encoder PID controller, motor driver, odometry, camera calibration, segmentation, path planning and path navigation. For each of these software modules were tested while being developed.

For the PID controller tuning was accomplished using the Robotet RoboRun Utility provided by the motor controller. The PID was properly tuned when the input and output of the motor speed were exact and was very reactive.

The motor driver was tested by achieving the ability to give the motors command velocities through ROS and also to receive messages from the motor controller.

Odometry took a while to test. For Odometry we used sensor fusion of the encoders and the two IMUs each stored on the LiDAR and GPS. We were able to visually test the accuracy of odometry through RViz, a python visualization library, and by studying the performance of our path navigation node.

Our path navigation node would take the odometry and a list of points and would output command velocities to allow the robot to travel through said points. The path navigation node algorithm was tested in a simulated environment with a custom made 3D model of the robot. Once the algorithm was at an acceptable stage path navigation node would be tested on the robot by having the robot travel around the perimeter of a square.

Camera calibration and segmentation were tested by observation. If the calibration and segmentation looked reasonable (was not warped or distorted) and was correctly proportioned then they were considered correct.

With segmentation path planning would use the probability map image (probability of what is and is not path) to create a line of points to reach the destination area in the image. Path planning was tested by running the whole stack while viewing the generated paths from the path planning node. Based on the autonomous movement of the robot the path planning algorithm would be adjusted. There we many methods used to try and improve our path planning capabilities such as using triangulation and then Dijkstra's algorithm. While using Dijkstra’s algorithm the “sink” (destination point) was changed from the farthest point, to the leftmost point, to other variations. Other methods were tried by reducing path sight and traveling the middle of the most visible path. These methods faulted due to camera to segmentation latency and would usually cause oscillations or slow reaction times for the robot. The largest area of improvement for the navigation stack is path planning.

Appendix

All source code is located in our repository here: https://github.com/romleiaj/pacman_ws

Or zipped in a folder parallel to this document.

Hardware:

Shown in figure 1 below is a circuit diagram of all our electrical, sensors, and high level equipment.

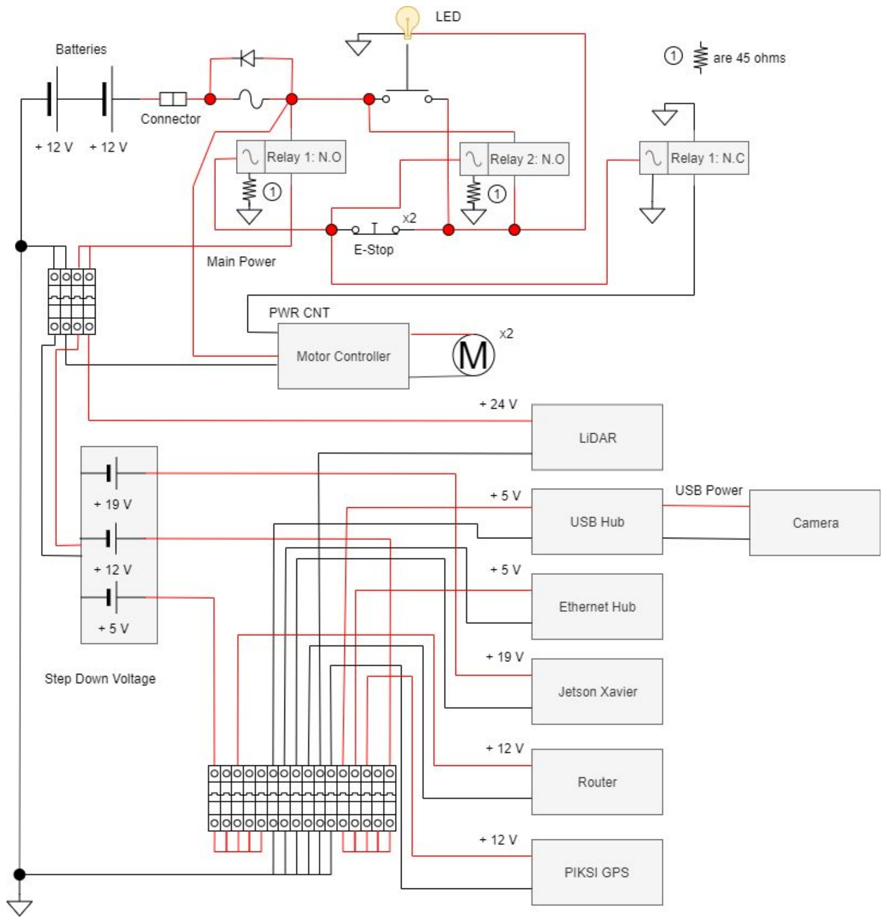

Figure 1. *System Power Block Diagram*

The high level components and sensors communicate through ethernet and USB. Shown in figure 2 below is the network connections between these components.

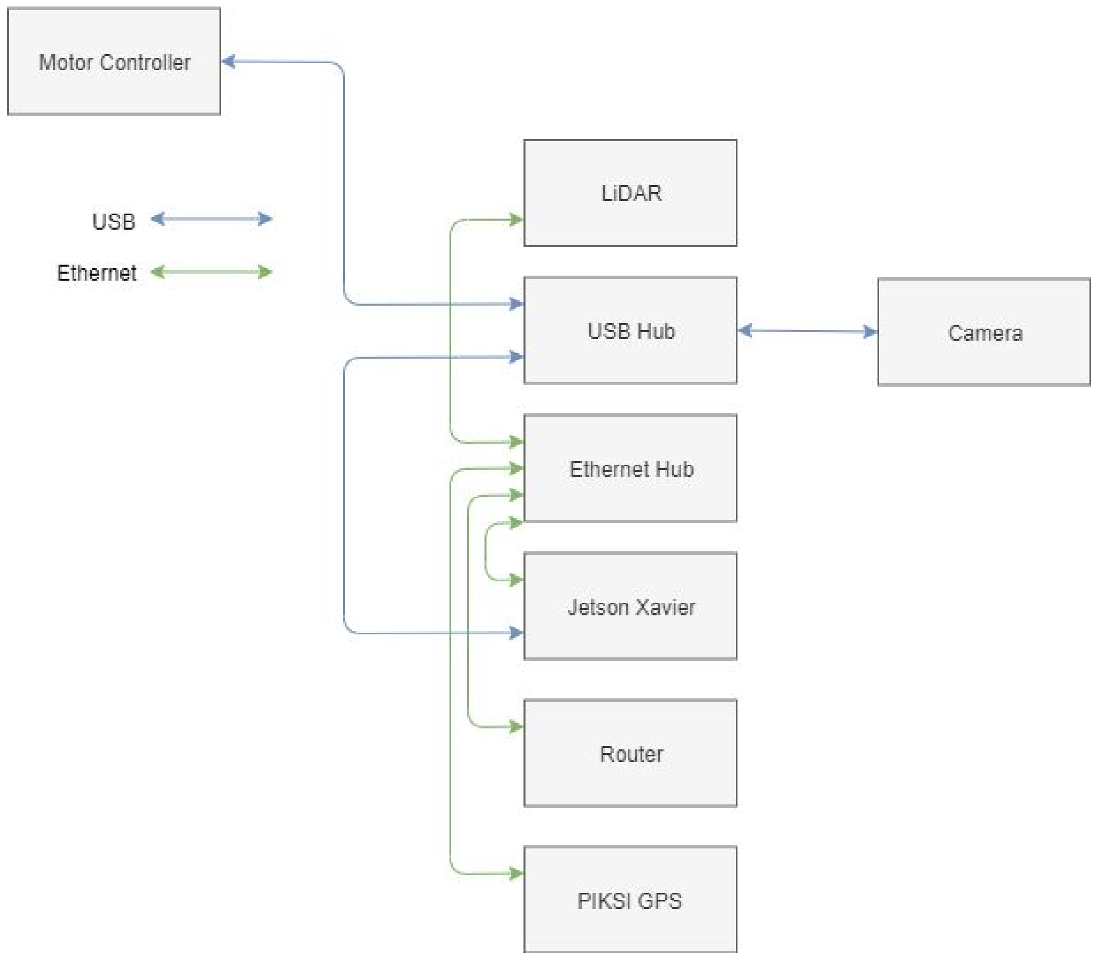

Figure 2. *System Hardware Block Diagram*