Proceedings of The National Conference
On Undergraduate Research (NCUR) 2016
University of North Carolina Asheville
Asheville, North Carolina
April 7-9, 2016

MANTRA Autonomous Agricultural Services Vehicle

Sean Finnigan, Daniel Casciato, Max Kern, Daniel Miller, Justin Alabaster, Gabriel Earley,
Benjamin Wagner, Guerin Williams
Mechatronics Engineering
The University of North Carolina Asheville
One University Heights
Asheville, North Carolina 28804 USA

Faculty Advisors: Neil Rosenberg and Susan Reiser

Abstract

Intelligent vehicle solutions are getting a lot of press lately, with autonomous vehicle technology at the forefront of many newsworthy tech stories. Many of these stories focus on the impending flood of expensive consumer products soon entering the market. Until recently, machine vision based autonomy was out of reach for most people/organizations interested in simultaneous localization and mapping (SLAM) problems. Lately however, sensor affordability and open source software resources have made accessibility a reality. This paper describes a simple, affordable technique for solving a mobile robot SLAM problem. This research shows that by using open source software, companies and individuals alike can affordably access these technologies. Our team created an autonomous agricultural services vehicle for ARGO ATV, a company that manufactures amphibious electric vehicles. In exchange for a donated chassis, ARGO asked us to design a system that can safely navigate and map a banana orchard (for applications in Martinique) without direct user input. This end was achieved by leveraging the resources of the Robot Operating System (ROS) and through collaboration with the open source robotics community. The system integrates information from a high resolution stereo vision camera, 2D LIDAR, Inertial Measurement Unit, and a GPS module to implement a real-time SLAM (Simultaneous Localization and Mapping) algorithm on a high performance, low power Embedded System. The vehicle can navigate autonomously while safely avoiding obstacles, animals, and farm workers.

Keywords: SLAM, ROS, Autonomous Navigation

1. Introduction

Autonomous vehicle navigation is a varied and robust area of robotic research. The introduction of open source software packages and affordable sensors allow for the low cost development of autonomous navigation systems. This project leverages open source technology to produce a competitive autonomous navigation solution capable of navigating a banana field in Martinique without user input. The proof of concept robot will be tested on a simulated field on the UNC Asheville campus.

2. Robot Operating System (ROS)

Robot Operating System (ROS) is an open source Linux-based software platform that is broad in scope and ability[3]. ROS is middleware, meaning that it is the middle ground between hardware and software communication through the use of nodes and topics. A ROS node is created for every sensor on a robot and serves as an information buffer so source code is not required to sort through serial data to find what is needed. The node takes the serial data and gets

valuable information then "publishes" that information to a topic. A topic is a way for nodes to have access to information that other nodes have been publishing. So if one node needs information from another node it will simply "subscribe" to the topic that has the information that the node needs. This can be a bit confusing, so to put everything into perspective ROS can be thought of as "the Internet," nodes can be thought of as "computers," and topics can be thought of as "URL's." Moreover, the "computers" publish information to the "URL's" on the "Internet." If another "computer" wants information from another "computer" it would simply go to the "URL" it needs and get that information.

ROS is the integration platform that takes in sensor data, runs it through control algorithms, then creates specified output. This means that all the sensors are "talking" directly with ROS making all sensor data easily accessible to the user. Creating a node for each task such as sensor data, control algorithms, or motor output creates a modular project interface that eases troubleshooting and lessens the computational burden on the designer. ROS also provides native diagnostic tools which eases debugging of the system. Ultimately, all of the nodes in use will filter down to a single node that receives a velocity command based on all of the processed data from the sensors. This node will read that velocity command and move the platform as specified.

The specific ROS nodes used include Rtab-MAP to accomplish visual odometry and mapping, Robot_Localization to deal with sensor fusion of all inputs, and Move_base which takes all this information and moves the platform accordingly. These software packages are the basis of this project, however they are only a few of the many software packages needed to integrate into the system.

2.1 RTAB-Map

This software package takes in data from a stereo vision camera to perform several SLAM algorithms creating 3 dimensional point clouds and occupancy grids. Rtab-map is great for mapping 3 dimensional space and localizing itself within that space. As it estimates the overall position of the platform within the map, it also checks for "loop closures" which means it looks for places that have been seen before and adjusts the map accordingly to eliminate any drift.

2.2 Robot_Localization

This package performs sensor fusion of all the different sensors within a system. An Extended Kalman Filter (EKF) is used to analyze each type of sensor and its accuracy[4]. The most accurate sensors hold the most weight in the estimation thus allowing the system to rely on the most accurate data. The estimated position read by each sensor is averaged together based on the EKF covariance of each sensor in order to obtain the most accurate representation possible with the given inputs.

2.3 Move_Base

This ROS node receives a goal and attempts to reach that goal based on the provided local and global cost maps. A cost map is a map that tells where obstacles are located near the platform (local) and everywhere the platform has seen obstacles (global). Knowing where obstacles are, having been identified by onboard sensors, Move_base creates a path to the received goal that avoids all obstacles.

3. Hardware

In creating an affordable SLAM platform one must be diligent in the choice of hardware, including sensors, a vehicle to control, and computers that integrate information taken from sensors into control algorithms. Choosing hardware that was affordable, reliable, and could stand up to the rigors of autonomous movement proved to be a challenging but achievable task.

3.1 Sensors

Four sensors are being utilized for this application: a stereoscopic camera, a laser range finder, an inertial measurement unit (IMU) and a GPS receiver.

The Stereolabs ZED stereoscopic camera tackles key issues faced by stereo vision users such as the tasks of camera calibration, and efficient depth computation. The ZED camera communicates directly through their software development kit (SDK) which offloads the depth of field computation onto an external graphical processing unit. A ROS wrapper pipes the data from the SDK and publishes it on to their respective ROS topics (depth image, rgb image, camera info, etc). This advanced design frees the CPU for other mapping, navigation and localization algorithm computation.

With any autonomous system, safety is the first priority. The goal with this system is to identify potential hazards and put the system on hold till the area is safe to navigate again. A laser range finder, or LIDAR, uses the reflection of an infrared laser off an object, and the timing between transmission and reception to quantify the LIDAR's distance from said object. These sensors are traditionally very cost intensive, but recent advancements and interest in the SLAM field have unearthed more cost effective options. The RPLidar manufactured by RoboPeak was both cheap and had existing ROS integration packages.

This sensor provides a 360 degree laser scan with a detection range just over 6 meters. The sensor returns 2000 (5.5hz) samples a second, of both distance and object angle that is then used to produce a 2D point cloud. This data is then used for mapping, localization, and object/environment modeling. The ROS node publishes the laser scan data to be integrated into the mapping system.

Outdoor testing of the RPLidar quickly revealed that direct and reflected infrared solar radiation caused critical interference. Whenever solar radiation was incident on the LIDAR unit's detector the sensor returned false objects at zero distance. This effectively tricked the system into falsely recognizing an object directly next to it. This solar interference is due to the LIDAR detector's sensitivity band including the frequencies of both the LIDAR's infrared laser and the infrared radiation from the sun. An infrared band pass filter was used to exclude the interference from the sun while allowing the LIDAR's own laser to pass through[1]. With this filter in place, further testing showed that the LIDAR now functioned as desired.

The Razor Inertial Measurement Unit (IMU) was sourced from Sparkfun electronics. This unit was selected because it incorporates three sensors - an ITG-3200 triple-axis gyroscope, ADXL345 triple-axis accelerometer, and HMC58883L triple-axis magnetometer effectively creating nine degrees of inertial measurement. The outputs of all sensors are processed by an on-board ATmega328 and output over a serial communication interface.

During initial sensor selection process the Piksi RTK GPS from Swift Navigation appeared to be the best choice due to an advertised 1cm level of accuracy and its small form factor, fast solution update rate, and low power consumption. The software architecture is open-source and had existing ROS compatibility. However, achieving a GPS lock proved almost impossible therefore it was replaced with the Ultimate GPS Breakout Kit from Adafruit. Even though this unit only provides accuracy to within 2-3 meters, implementation was simple and the desired output was achieved in one testing session. The system's inertial and vision navigation are robust enough for local positioning, and the GPS signal will be used for large area localization.

3.2 Integrated Computing

The nature of the ROS platform allows for different nodes to be run on separate computer hardware. To achieve a small form factor and fulfill computational power requirements two platforms were selected. The Jetson TX1 has an integrated NVidia graphics processing unit to process raw data from the Zed machine vision camera and LIDAR. An Odroid processes data from the IMU and GPS. The J5 chassis has an onboard computer that receives movement commands from the other two processors.

3.3 J5 Amphibious Vehicle

The Argo J5 is a four-wheeled electrically powered multipurpose vehicle. The vehicle has multiple ways of accepting control signals, however for this design, a wireless control will be implemented. The vehicle is capable of navigating almost any outdoor terrain, which makes it a perfect choice for this application. The vehicle has an on board Intel machine with a prebuilt ROS node that accepts velocity commands and communicates to the motor controllers.

3.4 Kinematics

The J-5 employs a skid-steer drive train in which some wheels slip while turning. This makes analytical estimation of vehicle pose difficult during turning maneuvers. A transformation to the more easily analyzed differential drive

configuration may be made by first taking measurements to determine the instantaneous center of rotation of the skid-steer chassis, and then determining the equivalent wheelbase of a differential drive chassis[2].

3.5 Power Supply

The power for the sensors, computers, and wireless router is supplied by two 12-volt lead-acid batteries wired in series. The voltage is stepped down to 12-volts and 5-volts via two voltage regulators and sent to a Cisco Wireless Router, Jetson TX-1, and Anker 4-port powered usb hub. The usb hub supplies power to the entire sensor stack as well as the Odroid.

4. Path Planning (Banana Nav)

The path-planning has two main tasks. Based on sensor data, the platform needs to maneuver through a field of trees until it has reached the end or is told to stop. It also needs to travel back to its starting position based on data obtained from mapping. The latter needs to be accomplished via the quickest route. For example, making a return to the charging station located at its origin, the platform would need to take the shortest path back.

4.1 Unity Simulation

Unity game engine is a popular game creation platform that has been repurposed to create a simulation environment for this system. Unity has a built in realistic physics engine and an expansive developer community. The environment spawns the field of trees and internal functions simulate LIDAR and IMU data. The system uses this sensor data to move a mock car through a field without hitting any trees. This simulation is used to develop and refine the path planning algorithm in absence of a physical vehicle and sensors.

4.2 Algorithm

To accomplish autonomous navigation Move_base and a custom control node called Banana_Nav were implemented. Based off of provided sensor data, Banana_Nav determines where the platform should go next and sets a goal. Then, Move_base creates a safe path to the goal and sends the correct velocity values to the motors of the platform. Trees are used as waypoints to implement the path planning mathematics. The system identifies the midpoint between two trees in front of the vehicle and set that as the goal. This algorithm also contains the means of identifying the end of a single row of trees and the end of a field of trees to accomplish full autonomous navigation.

4.3 Implementation

The Banana_Nav algorithm is implemented by reading an occupancy grid from Move_base. An occupancy grid is an array of 8-bit integers. Each integer represents a cell on a cost map. The closer the value is to 100 the more likely that cell has an obstacle in it. Move_base creates this grid based off the sensor data it receives. The algorithm then reads the data through a topic published by Move_base. Once Banana_Nav has the occupancy grid, it begins by traversing the first row. The assumption is that the user starts the vehicle in front of a row of trees. It locates the first two obstacles (the first to its right and first to its left) and sets the goal as the midpoint between them. It sends the goal to Move_base through an action server, which is similar to a ROS topic. The difference is that the action server can be both a publisher and a subscriber. So a goal is sent and then receives a success or failure signal though the same channel. After the first goal is sent, the algorithm continues to send midpoint goals until it sees no trees in front of it. It then moves on to the next row and begins to go down it. This process is repeated until there are no more rows. When it is done navigating, it sets the goal to return to the place it was turned on. Move_base then plans the fastest route returns it to its starting position.

4.4 Gazebo Simulation

In cases in which real world testing was impossible a simulation environment was used to replicate the ideal conditions needed to test the platform. Cases where the simulation was preferred over real world testing included rainy days or

at night when the sensors could not give accurate readings. The simulation environment that was chosen was the gazebosim simulator created by the Open Source Robotics Foundation. It allowed for the accurate simulation of the platform and provided accurate sensor data that could be used to test the ROS nodes used by the system.

5. Product Design

In designing the physical appearance of the system a method known as "The networked world approach" was implemented[5]. According to the authors of Breaking Smart, "The process is not even recognizable as a problem-solving mechanism at first glance." It looks like this:

1. *Immersion* in relevant streams of ideas, people and free capabilities
2. *Experimentation* to uncover *new* possibilities through trial and error
3. *Leverage* to double down on whatever works unexpectedly well

Our team, knowing that we wanted to work on a project focused on machine vision, immersed ourselves in the appropriate literature and online resources. We experimented heavily with ORB Slam, LSD Slam, and Google's Project Tango. We played with webcams and drone-mounted cameras, smart phones, and laptop cameras. This process, along with our implementation of a team communication app called Slack, opened up a thriving ecosystem of open-ended tinkering for our team. Constant communication and the serendipity of unfolding events was empowering and fun. Once we found what worked unexpectedly well, we kept at it.

Once we solidified sponsors, we began to narrow our focus, but did not lose the freedom to tinker. Our evolving product took on many appearances, definitions, and applications, while maintaining our core interests: computer vision, product design, and fun.

5.1 Mechanical Design

We designed an elegant sensor housing for several reasons. First of all, we wanted the experience of designing an actual, physical product. The fabricated housing appeals to prospective end users and provides us with a modular solution, mountable on any vehicle. This is important because the orientation of the sensors is an integral part of the ROS-based navigation and sensor fusion system. Although, for our specific vehicle, we had to insure that the vision-based sensor systems, the stereo camera, and the LIDAR would not be obstructed by the body of the robot. Therefore, a sensor housing allows for a portable and repeatable sensor system that eliminates any physical interference from the vehicle being manipulated.

5.2 Manufacturing Processes

In the interest of having an industry-standard manufacturing experience, we chose to build with several different materials. We selected machined and water-jetted aluminum for the base plate, because it is lightweight, stiff, and easy to manufacture. Carbon fiber was chosen for the cowling, because it demands that we make a mold and learn how to work with carbon fiber. We opted to use some 3D printed material because we can make it fast and cheaply. The process of providing technical documentation to our vendors was an added bonus for team members who had not worked with geometric dimensioning and tolerancing (GD&T) vendor documentation in the past. Additionally, our team learned how to build cables, source parts, interface with sponsors and the public, and produce a product that is elegantly designed.

6. Conclusion

Autonomous vehicle navigation is an exciting technology that continues to find new and useful outlets. In the past, this technology has been cost-prohibitive, affordable only by select industries and individuals. The increasing availability of cheap and reliable sensors and open source software packages has driven down costs, and in turn, the popularity of the technology has resulted in myriad hardware and software developments by a community excited by the spectrum of possible applications. The widespread availability and affordability of this technology precipitates new applications. For this application, the use of the Robot Operating system, stereo vision camera, imu, gps, and

LIDAR allowed for the development of an autonomous navigation hardware and software system that can compete against other available products in both function and cost. The proof of concept navigation system created with these hardware and software elements proved successful in traversing a simulated banana field.

7. Acknowledgements

We would like to thank the organizations that aided us both financially and intellectually: ARGO ATV, AMS Advanced Manufacturing Solutions, AVL Technologies, Google, Nvidia, Open CV, Open Source Robotics Foundation, pointcloudlibrary, Robot Operating System, Textron, and Ubuntu.

8. References

1. Fredell, Markus A. 2015. *Sub-nanometer band pass coatings for LIDAR and astronomy* . Accessed March 1, 2016. https://www.omegafilters.com/documents/file/Sub-Nanometer-Band-Pass-Coatings-SPIE-96120K.pdf.
2. Mandow, Anthony. 2014. *Experimental kinematics for wheeled skid-steer mobile robots.* Accessed February 12, 2016. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.565.755&rep=rep1&type=pdf.
3. Quigley, Morgan. 2009. *ROS: an open-source Robot Operating System.* Accessed March 2, 2016. https://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf.
4. Ribeiro, Maria Isabel. 2004. *Kalman and Extended Kalman Filters:*. February. Accessed March 1, 2016. http://users.isr.ist.utl.pt/~mir/pub/kalman.pdf.
5. 2013. *Tinkering vs goals.* Accessed January 30, 2016. http://breakingsmart.com/season-1/tinkering-versus-goals/.