Embedded Mobile ROS Platform for SLAM Application with RGB-D Cameras

Andrew Newman, Guojun Yang, Boyang Wang, David Arnold and Jafar Saniie

Embedded Computing and Signal Processing (ECASP) Research Laboratory (http://ecasp.ece.iit.edu/)
Department of Electrical and Computer Engineering
Illinois Institute of Technology, Chicago, IL, U.S.A.

***Abstract*—Mapping and localization are essential for various robotic applications. Without a GPS signal, indoor navigation remains a challenging task. With recent progress in machine learning, embedded systems, and sensor technologies it has become practical to use computer vision algorithms to perform localization and mapping tasks on low power, portable robots. The focus of this project was to create a tracked vehicle with autonomous navigation capabilities for use in traversing difficult terrain. Such an unpredictable environment posing additional challenges to our navigation system. Navigation is achieved via processing source stereo image provided by an Intel RealSense D435i depth camera using the NVidia Jetson Nano, a CUDA-enabled single-board computer. The design heavily relies on the usage of the Robot Operating System (ROS) and Simultaneous Location and Mapping (SLAM) algorithms to enable flexibility in future feature expansion and ease of communication.**

Keywords—Autonomous Navigation, Simultaneous Localization and Mapping (SLAM), Robotics Operating System (ROS), NVidia Jetson Nano

I. INTRODUCTION

As the research of autonomous vehicles has become prevalent in the present day, the focus has shifted towards the feature expansion of autonomous vehicle capabilities in the areas of navigation algorithms, addition of sensors, and image processing. However, the majority of autonomous vehicle projects tend to focus on its application in highly controlled areas such as warehouses and road infrastructure, relying on external markings such as road lane indicators to assist navigation [1].

Relying on external markers limits a robot's location of operation. This includes research studying indoor localization using makers or patterns. These methods share one common disadvantage: they require modification of the environment. Ideally, a robot traversing a space should be able to orient itself while adapting to changes in the environment. With progress in Artificial Intelligence (AI) and embedded systems, the possibility of using a robot to navigate in a generic environment arises [2].

In order to navigate without supervision, a robot needs to be able to sense its surrounding environment and respond. Such a process requires the robot to be equipped with the proper sensor(s) and gain situation awareness through algorithms in real-time. To sense the environments, the use of cameras, laser rangers, and ultrasonic sensors can be employed. Ultrasonic sensors and laser rangers are ideal for measuring distance; however, they are not ideal for extracting geometric information. Cameras can extract geometric and texture information; however, they are not ideal for measuring distance. To resolve these difficulties, we employed an RGB-D camera in this project.

To map the environment using the data captured by the RGB-D camera, the system needs to recover the geometry of space by stitching the captured data in real-time. In this project, mapping and navigation require the system to maintain a sense of orientation and record key objects' positions. With the camera and accelerometer equipped, these problems can be resolved using SLAM (Simultaneous Location and Mapping). SLAM is a computational heavy algorithm, but with GPU acceleration it can run in real-time on embedded devices.

This project intends to focus on the implementation of an off-road autonomous vehicle to broaden knowledge surrounding the implementation of such a project. Generally, off-road vehicles tend to exhibit a higher degree of robustness and reliability of operation as compared to road vehicles. To that extent, off-road vehicles tend to utilize either low-pressure tires with deep profiles, tracked transmission, or some combination of both. A high level of driver training is required as well since navigation is a very major factor in the success rate of traversing difficult terrain. All source codes are made available on GitHub: *https://github.com/Nemiland/ECE-597-Automonous-Robot*

II. RGB-D MAPPING USING SLAM

The major component that enables this system to function is a SLAM algorithm. Its primary function in the system is to organize and position the incoming sensory data in order to build and assess an environment map. The crux of the SLAM algorithm is to compare the change in a captured photo (data) for determining the motion of the camera as well as the geometry of the scene. To map a scene, the following steps are performed: feature extraction; feature matching; geometric reconstruction; and bundle adjustment.

A feature detector extracts key points from an image by analyzing the gradient change at a location. With gradient change in multiple directions, such as a corner, a feature can be detected; then the same feature can be associated in other frames. Features from these images are usually scaled and orientation variant, hence the same process needs to be performed on multiple scales of the same image. Unlike traditional SIFT and SURF features, ORB performs feature extraction and descriptor calculations much quicker [3].

978-1-7281-5317-9/20/$31.00 ©2020 IEEE

With Feature points detected in each frame, the algorithm needs to associate the same feature point across the scene in order to determine the geometric relation. The matching process relies on the feature descriptor, which records the gradient changes around each feature point. These gradient changes are encoded as a vector. Each vector will be normalized to ensure orientation invariance. Finally, if the Euclidian distance between two vectors is smaller than a threshold a match is found.

With enough feature points matched across scenes, the geometric relationship in the scene can be recovered. The SfM (Structure from Motion) algorithm uses the matched feature points to estimate the motion of the camera and the geometric information simultaneously [4]. Matched features can also be used for detecting loops in camera trajectory. An example of a video analyzed by a SLAM algorithm [5] is shown in Figure 1. The orange lines indicate a feature point matched between 2 adjacent frames.

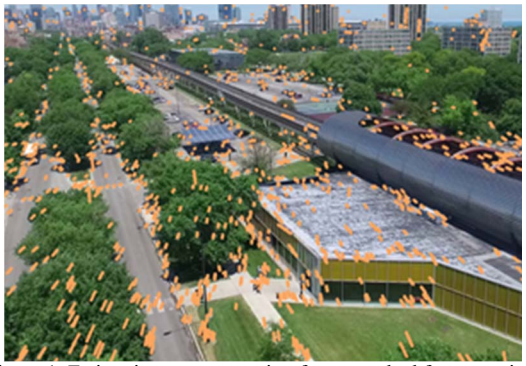

Figure 1. Estimating camera motion from matched feature points

In this particular application, the pursuit of navigation via 3D space localization was achieved using the ORB-SLAM2 algorithm [6]. The proposed algorithm structure rebuilds the depth point cloud stream generated from the depth camera over time domain and reconstructs it into a continuous map with points localized in their relative positions to one another. It was chosen due to its compatibility with Robot Operating System (ROS) [7], the capability to function in outdoor applications, and added RGB-D camera support as compared to its predecessor [8].

The distinguished high performance and reliability of the system can be accredited to its utilization of the ORB [3] feature matching descriptor in its pattern recognition subsystem as well as DBoW2-based [9] re-localization routine.

III. System Implementation

The proposed system is an autonomous, self-driving, tracked vehicle capable of navigating in a 3D space via utilization of depth maps generated via a stereo camera. The design of this project focuses on the scalability and flexibility of implementation to various platforms.

The system can be broken down into three major sub-systems: Computation and Cognition; Mobility and Actuation; Communication and Control. The relationships between these three sub-systems are shown in Figure 2. The communication and control sub-system is in charge of receiving commands from the user and can be used for direct control of the robot. In the event of an emergency, the user can take over control from the computer. The communication subsystem can also establish a wireless data link with a server that logs the activity of the robot. The mobility and actuation sub-system is in charge of driving the actuators of the robot, providing power to the rest parts of the robot. The mobility sub-system is also responsible for monitoring the status of the robot, such information is critical for maintaining the safe operation of the robot. The computation and cognition sub-system is the brain of the robot; it takes inputs from various sensors and uses its computational power to generate decisions.

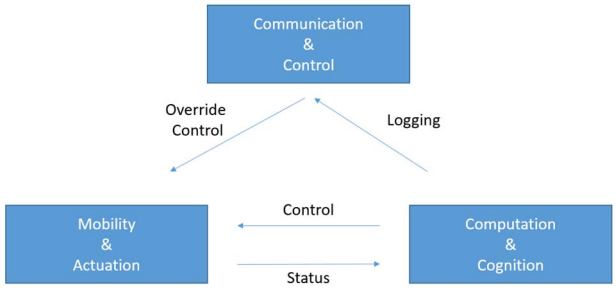

Figure 2. Information flow between subsystems

The prototype of the robot platform is shown in Figure 3. The system is built using the ROS framework, each sub-system operates as a node of the system. Specifically, ROS is installed on the Jetson Nano microcomputer. The cognition and computation system are comprised of the ORB-SLAM2 algorithm and RealSense 435Di, an RGB-D camera with IMU. The communication and control of the system are carried out by a Wi-Fi module on the Jetson and Bluetooth on the ESP32. The Mobility and actuation are achieved via the ESP32 and 4 DC motors (one for each track).

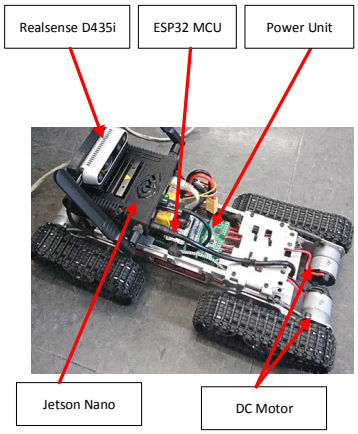

Figure 3. Assembled prototype platform

The system is arranged in a controlled workflow and is currently being implemented sequentially. First, the chassis, as well as the power system, are set up to accommodate for future modules, then the vehicle control is implemented via ESP32 microcontroller and Toshiba TB67H420FTG Dual H Bridge motor controllers.

As means of debugging as well as retaining the control of the robot in case of contingency, the external override control is implemented via Bluetooth communication to a terminal PC.

The majority of image processing and data manipulation is achieved via the utilization of NVidia Jetson Nano single-board computer. It is mounted atop the chassis using a custom bracket enclosure and is responsible for processing the IMU and RGB-D camera feed from Intel RealSense D435i. It also hosts the server application for ROS. ROS in this configuration is responsible for setting up a common communication platform with the ESP-32 microcontroller for motor control and the desktop PC for data collection and visualization, as well as establishing data format of all of the software subsystems. This is again done to allow the resultant system to be as modular as possible, with the ability to expand the platform to include multiple sensors or motor platforms. Figure 4 describes the overview of the system design flow.

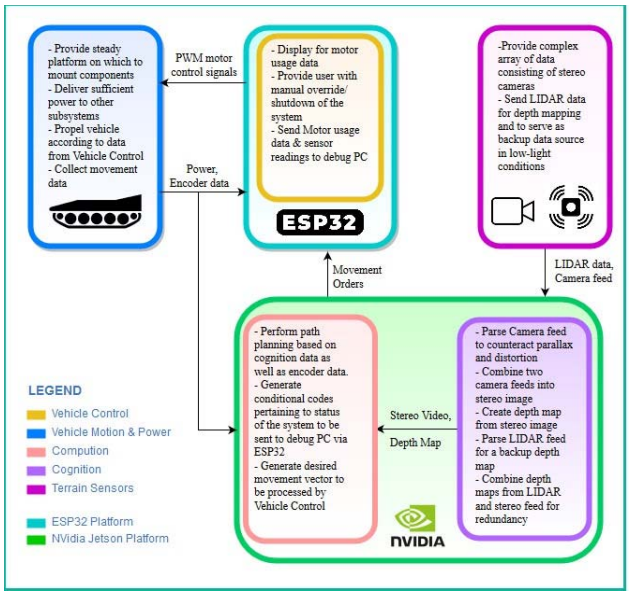

Figure 4. System Design Flow

Following is a list of tasks that have been completed as part of this project:

A. Chassis Assembly and Power System

The chassis arrived as a kit and required some assembly with minimal instructions provided. The wiring harness was implemented to accommodate two motor controller inputs as well as a step-down converter made out of car charger being wired to a Li-Po battery.

The logic components of the vehicle control subsystems required a twelve-bit bus between the ESP32 and logic converters as well as a logic converter to the motor controller. Additionally, every component needed a supplied power rail to function. The increasing complexity of the circuit yielded the poor durability when implemented on the breadboard with jumper cables and was hence implemented on the protoboard.

B. ESP32 Microcontroller

Since Espressif Systems, manufacturer of ESP-32 platform, has provided the support of the Arduino IDE via the inclusion of the machine code compiler module, Arduino IDE was the natural choice for implementation due to it having a large library selection for a multitude of tasks. Arduino IDE also has support for the ROSSerial: a serial standard for communicating with the ROS core process, functionally implementing the ESP32 as a ROS Node. The vehicle control code captures linear velocity as well as angular turn vector that is supplied either from the x and y axes of the game controller or the geometry *msgs::Twist* message format that is prevalent for robot control inside ROS. It then uses these vectors and adjusts the actuation of motors on the left and right side of chassis according to the paradigm of differential drive robot control. The system override was one of the desired features of this project to prevent the autonomous vehicle from catastrophic failure when human supervision is available, hence a Bluetooth server program was implemented in Python 3 utilizing PyGame library for game controller input support as well as the PyBluez library for the Bluetooth interface to the ESP32 board.

C. ROS Communication Setup using Jetson TX2

The next step in implementing a self-driving vehicle is establishing a framework inside ROS that would allow the messages from one node directly to affect the vehicle controls inside the ESP32 ROSSerial node. For this state, the connection between Jetson TX2 and ESP32 is implemented via the Soft Access Point Wi-Fi located on the ESP32. An ESP32 node is integrated into the ROS runtime via the serial node library and is then being controlled using a program generating Twist format controls. The selected test software of choice is *teleop_twist_keyboard*, which allows the user to manually change linear and angular velocity vectors sent to the ESP32. This allows the user to manually test every transition case of the communication for ensuring no bugs surface at this stage in production.

Finally, the choice was made to transition the system to use the NVidia Jetson Nano platform, a newer edition to the Jetson family of products that boasts a smaller overall footprint while offering reasonably high performance. The Jetson Nano does not come with the Wi-Fi module built-in, so to provide wireless connectivity, an Intel AC9200 series Wi-Fi card was introduced to the system. Additional mounting has been provided for the Intel RealSense D435 depth camera.

D. RealSense RGB-D Feed Integration into ROS

Since one of the design goals of this project was to perform autonomous navigation in ROS using a stereo camera, we needed a depth camera to operate within the ROS messaging system. Our choice for the project was a RealSense D435i Depth Camera. This camera module, developed by Intel Corporation, offers several features that are essential for efficient video processing. Firstly, it offers synchronization and basic calibration of the camera feeds as well as computation of point cloud via utilization of onboard Intel RealSense D4 Vision Processor. It bolsters an RGB video output of 1920x1080@30fps and depth stereo image output at 1280x720@90fps, which allows for fast refresh rates essential in real-time navigation while reserving higher quality footage for any post-processing on physical image feed. Lastly, it features a very formidable outdoor performance on its IR emitter module; the camera can perceive distances for its point cloud computation in a range of 1-15 meters.

E. *Integrating a Visualizer to Represent SLAM data*

The way SLAM algorithms operating in ROS represent spatial data that is largely leveraged by the use of the Transform Frames (TF) [10]. On startup and first data collection, the algorithm generates a base map transform, which acts as the origin point onto which all subsequent data is projected onto. When the algorithm adds new data, it then extrapolates its relative position to map frame through IMU location data as well as previously perceived landmarks to add cross-referential positional data. This allows the platform to stitch multiple video frames into a complex 3D environment onto which processing can be added. This is mostly because we now have enough data to implement any kind of automation algorithm that we would desire, be it map exploration, or scanning for foreign/missing objects, or additional metadata collection.

IV. RESULTS

To assess the platform, a series of experiments were carried out. RealSense video camera feed integration into the ROS ecosystem is shown in Figure 5. Since one of the design goals of this project was to perform autonomous navigation in ROS using a stereo camera, we needed a depth camera to operate within the ROS messaging system.

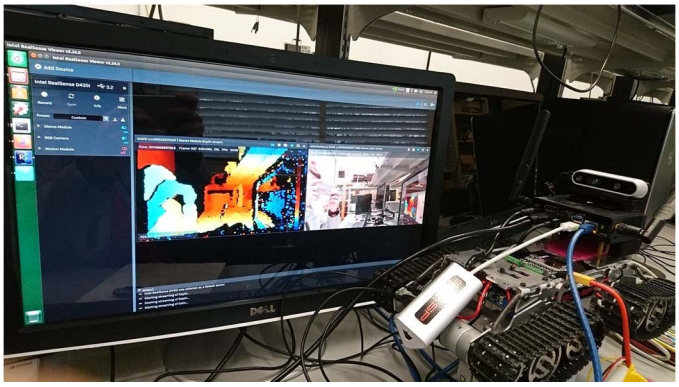

Figure 5. Video feed processing performed on Jetson Nano

As shown in Figure 6, we have performance optimizations and ROS Multiple Machines support. During stress testing of the system, a platform flaw manifested through an insufficient amount of resources available for processing. While Jetson Nano boasts an impressive amount of computational power, it lacks in regard to the amount of memory available for processes. This had a huge impact on the SLAM map generation, as large maps would cause the entire system to stall due to an insufficient amount of RAM.

A countermeasure to this issue was implemented in the form of establishing a network link over to one of the workstation computers onto which the ROSCore and SLAM algorithm storage was placed. ROSCore acts as the host process to the entire messaging network implemented in ROS and boasts the capability of having external slave machines connecting to it and using the messaging platform remotely. As a result, after configuring the Jetson Nano to connect to the workstation via Ethernet and configuring the ROS structure to accommodate network transfer of the processed video from the depth camera as well as the IMU localization data.

As shown in Figure 7, the ORB-SLAM2 allowed for mapping to be implemented without the IMU onboard due to its ability to detect rotations based on shared landmark features of consecutive frames, however, its inflexibility in the application, as well as lack of technical documentation, have made it difficult to implement it in this specific project.

V. CONCLUSION

The result of this research project is a platform capable of performing complex computational processes in areas of image processing and navigation path generation while operating completely independent with internal battery array and wireless diagnostics and control override assisting the functionality of the vehicle.

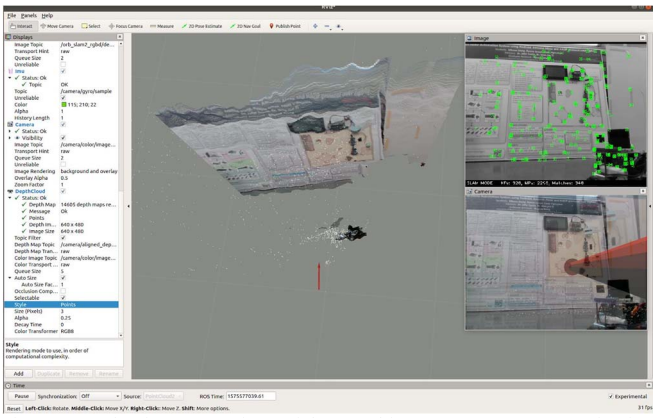

Figure 6. Projection comparison of the map history versus current projection

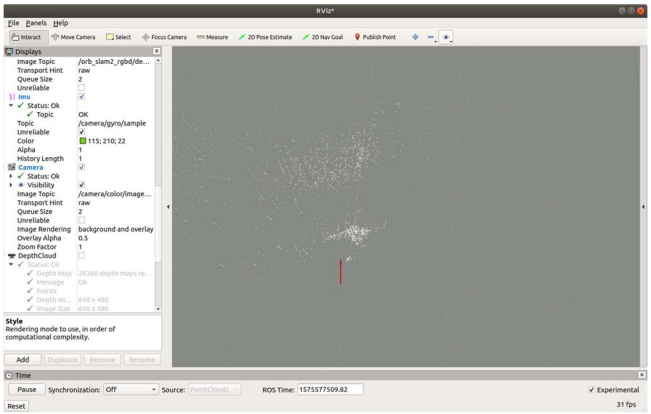

Figure 7. Sample point cloud generated by ORB-SLAM2

REFERENCES

[1] J. Huh, K. Lee, W. K. Chung, W. S. Jeong and K. K. Kim, "Mobile Robot Exploration in Indoor Environment Using Topological Structure with Invisible Barcode," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, 2006.

[2] A. Kouris and C. Bouganis, "Learning to Fly by MySelf: A Self-Supervised CNN-Based Approach for Autonomous Navigation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, 2018.

[3] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: an efficient alternative to SIFT or SURF," *International Conference on Computer Vision,* 6 November 2011.

[4] G. Yang, Z. Zhou, T. Gonnot and J. Saniie, "Design flow of motion based single camera 3D mapping," in *2015 IEEE International Conference on Electro/Information Technology (EIT)*, 2015.

[5] G. Yang and J. Saniie, "Design Flow of Single Camera Motion Estimation Using GPU Accelerators," in *2019 IEEE International Conference on Electro Information Technology (EIT)*, Brookings, 2019.

[6] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: an Open-Source SLAM System forMonocular, Stereo and RGB-D Cameras," *IEEE Transactions on Robotics 33.5,* pp. 1255-1262, 12 June 2017.

[7] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Ng, "ROS: an open-source Robot Operating System," *ICRA workshop on open source software,* p. 5, 12 May 2009.

[8] R. Mur-Artal, J. M. M. Montiel and J. D. Tardos, "ORB-SLAM: a Versatile and AccurateMonocular SLAM System," *IEEE transactions on robotics 31.5,* pp. 1147-1163, 24 August 2015.

[9] D. Galvez-Lopez and J. D. Tardos, "Bags of Binary Words for Fast Place Recognition inImage Sequences," *IEEE Transactions on Robotics 28.5,* pp. 1188-1197, 18 May 2012.

[10] "ROS 3D Robot Visualizer," [Online]. Available: https://github.com/ros-visualization/rviz. [Accessed 9 February 2020].