Technical White Paper

Autonomous Mobile Robots: Toward Intelligent and Safe Navigation With AM69A

Do-Kyong Kwon

ABSTRACT

Autonomous navigation of a mobile robot requires high-performance computing resources of both Artificial Intelligence (AI) and traditional computer vision to perform various tasks with multiple sensors. In this paper, the main tasks and related technologies for the autonomous navigation of mobile robots are introduced. This paper also demonstrates that the AM69A processor, which is the high-end device among the AM6xA scalable embedded processor family, is well designed for the autonomous navigation in terms of the performance and ease of development.

Table of Contents

List of Figures

List of Tables

1 Introduction

Autonomous Mobile Robots (AMRs) help improve productivity and operation efficiency in manufacturing, warehousing, logistics, and so forth. For example, AMRs can carry packages in warehouses and logistic centers, vacuum-clean floors, and serve foods and drinks in restaurants. Early AMRs used to operate in workspaces restricted from humans and navigated along the predetermined path guided by lanes and AprilTags on the ground. Therefore, early AMRs did not necessitate lots of sensors and stringent functional safety features. Such robots that follow the predefined path are also called automated guided vehicles (AGVs). On the contrary, recent AMRs are equipped with advanced sensors to operate in the shared workspace with humans and navigate freely but safely through the environment to perform assigned tasks at designated locations with as little human intervention as possible.

As shown in Figure 1-1, there are three main tasks for AMRs to self-navigate safely: localization, perception, and planning. First of all, the mobile robot must know their own location in the workspace. Accurate localization is the minimum requirement for autonomous navigation. Once localized, the mobile robot must perceive the dynamic environment with moving objects including humans and other robots in operation. Next, the robot must plan a path to the destination to control themselves accordingly to avoid situations that cause safety concerns. This paper discusses how these tasks work and the inherent challenges while focusing mainly on localization and mapping.

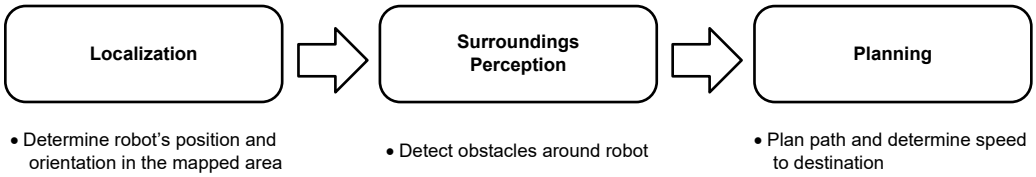

Figure 1-1. Three Main Tasks for Autonomous Navigation

The AM69A processor is a heterogeneous microprocessor built for high-performance computing applications with traditional analytics and AI algorithms. Key components include octal Arm® Cortex® A72 cores, Vision Processing Accelerator (VPAC), quad C7x Digital Signal Processor (DSP) with Matrix Multiplication Accelerator (MMA), Graphical Processing Unit (GPU), video codec, isolated MCU island, and so forth. VPAC has multiple accelerators including an Vision Imaging Subsystem (VISS), that is, Image Signal Processor (ISP), Lens Distortion Correction (LDC), and Multi-Scaler (MSC). Figure 1-2 illustrates the simplified AM69A block diagram. More details are found in the AM69x Processors, Silicon Revision 1.0 data sheet. Multi-camera AI use cases on AM69A are introduced in the *Advanced AI Vision Processing Using AM69A for Smart Camera Applications* technical white paper. This paper explains why AM69A is the best processor to run all three tasks simultaneously for autonomous navigation.

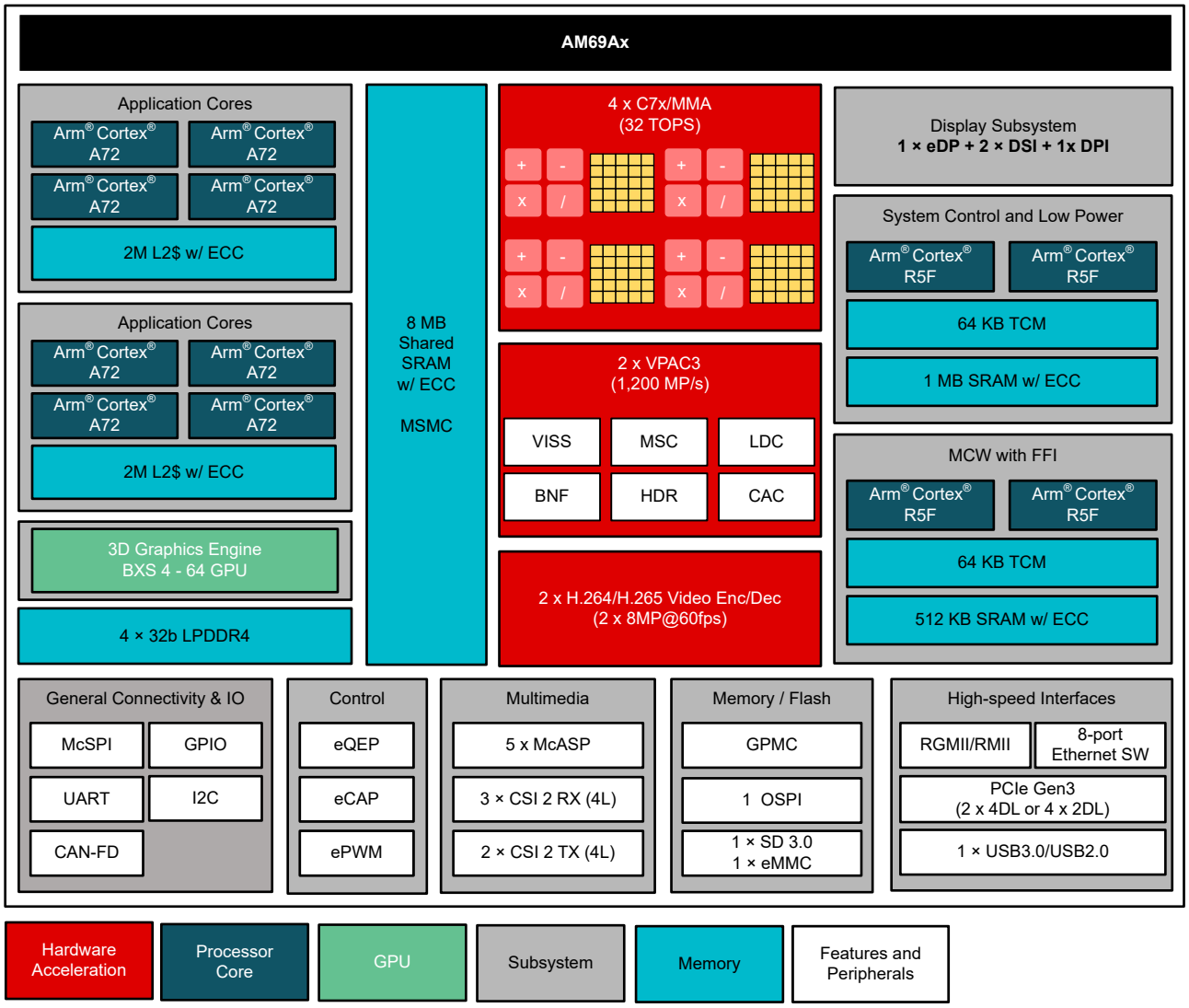

Figure 1-2. AM69A Simplified Block Diagram

2 Localization and Mapping

The process for making the robot navigate along the fixed path is easily accomplished with lanes and AprilTags painted on the ground. For example, the robot can be programmed to travel from one tag to another by defining what the robot must do upon detecting and recognizing each tag, for example, move forward, turn right and move, lift package and move forward, and so forth. In such scenarios, the robot does not need to localize as long as the robot can be controlled precisely as directed.

Localization is the process for the mobile robot to calculate their position and orientation in the mapped area while navigating dynamically. In most cases, the map is built in advance by the mobile robot as well and saved in the format that the robot can reuse for localization. The mobile robot explores an unknown environment and updates their location while building the map simultaneously. For this reason, this mapping process is called Simultaneous Localization and Mapping (SLAM).

2.1 Simultaneous Localization and Mapping

SLAM algorithms are grouped into three categories depending on the techniques used; filter-based SLAM, graph SLAM, and Deep Learning (DL) based SLAM. The filter-based SLAM treats the problem as a state estimation problem. The state, which comprises the pose and map of the robot, is updated by a filter iteratively based on measurements as the robot explores. The DL-based SLAM solves the problem by replacing an entire end-to-end process with DL networks. Only sub tasks in the graph SLAM can be replaced with DL networks. However, it

is reasonable to classify such algorithms as the graph SLAM. The graph SLAM is currently the state-of-the-art algorithm. The end-to-end DL-based SLAM does have recent promising results but is not mature, and the filter-based SLAM performs worse than the graph SLAM in general. Therefore, this paper limits the discussion to the graph SLAM. SLAM algorithms can be classified further based on the primary sensor, for example, visual SLAM, LiDAR (Light Detection and Ranging) SLAM, and so forth. Inertial Measurement Units (IMU) or Inertial Navigation Systems (INS) are often used together with the primary sensor to improve the accuracy of pose estimation.

2.2 Graph SLAM

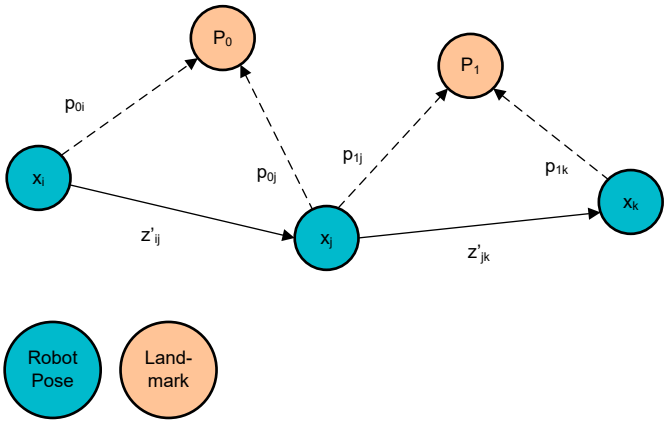

Figure 2-1. Graph With Poses of the Robot and Observations

The graph SLAM treats the SLAM problem as non-linear optimization with a graph structure, where nodes represent the poses of the robot and observations at different times and edges represent constraints between the poses. Figure 2-1 represents a toy example illustrating how the graph is built with the poses and observations of the robot as the robot moves. $\mathbf{x_i}$, $\mathbf{x_j}$, and $\mathbf{x_k}$ are the poses of the robot and $\mathbf{P_0}$ and $\mathbf{P_1}$ are the landmarks, objects, or points observed by camera. Here it is assumed that $\mathbf{P_0}$ is observed from $\mathbf{x_i}$ and $\mathbf{x_j}$ and $\mathbf{P_1}$ from $\mathbf{x_j}$ and $\mathbf{x_k}$. And $\mathbf{p_{nt}}$, where n = 0, 1 and t = i, j, k, is defined as the position of $\mathbf{P_n}$ when observed from $\mathbf{x_t}$. The goal of the graph SLAM is to find the transformations between the poses of the robot, that is, $\mathbf{z'_{ij}}$ and $\mathbf{z'_{jk}}$ so that the same observations from different poses overlap maximally. Therefore, determine $\mathbf{x_t}$, where t = i, j, k, so that $(\mathbf{p_{0i}} - \mathbf{p_{0j}})^2 + (\mathbf{p_{1j}} - \mathbf{p_{1k}})^2$ is minimized.

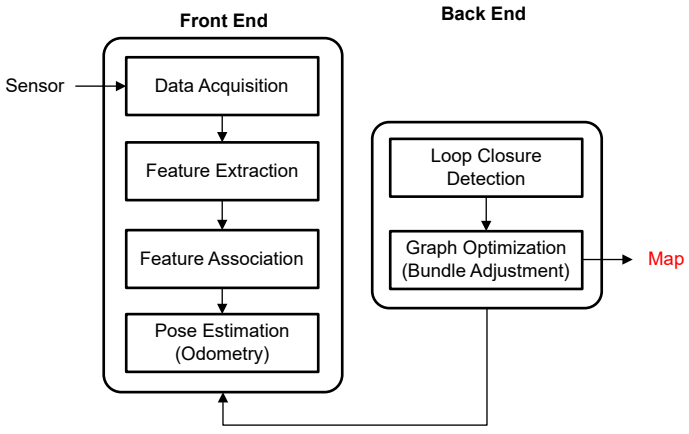

Figure 2-2. Graph SLAM Process

Figure 2-2 shows the graph SLAM process, which comprises two main components, the front end and the back end. The front end processes input sensor data to estimate the poses of the mobile robot and the distinctive features around the robot. The front-end processing consists of the following steps:

- Data acquisition: Data is captured continuously from sensors such as a camera and LiDAR. The captured sensor data is pre-processed here. For example, ISP, lens distortion correction, rectification of stereo image,

and motion compensation of LiDAR point cloud are performed during data acquisition. The captured data within a fixed time interval forms a frame.

- Feature extraction: Distinct and descriptive features are extracted from each frame. These features can be key points in images (visual SLAM) or geometric features such as edges, planes and scanned 2D or 3D points themselves (LiDAR SLAM).
- Feature association: The mobile robot associates the extracted features across different frames to determine which features correspond to the same ones in the environment.
- Pose estimation: Based on the associated features, the mobile robot estimates the changes of its position and orientation between two frames and therefore estimates the new pose of the mobile robot. Once the pose is estimated, the map can be updated using the associated features.

The poses of the mobile robot estimated in the front end are not error free and the errors are accumulated as the mobile robot traverses, which can result in huge drift errors. The back end is responsible for refining the estimated poses and updating the map. This refinement consists of the following steps:

- Loop closure detection: This is the process that determines if the current location has been visited before. Identifying the visited locations is critical in reducing drift error with graph optimization.
- Graph optimization: A graph is maintained in the graph SLAM as shown in Figure 2-1. The goal is to update the previously-estimated poses so that the observation, for example, features, from one pose maximally overlap with the same observation from other poses. Non-linear optimization techniques like Newton, Gauss-Newton, and Levenberg-Marquardt (LM) optimizations are used for this purpose.

In general, the created map is a set of the extracted features with their positions and descriptors in visual SLAM and a set of geometric features or points themselves in LiDAR SLAM. In many use cases, the map is also saved in the form of occupancy grid map after post-processing to make obstacle detection and path planning easier.

2.3 Localization

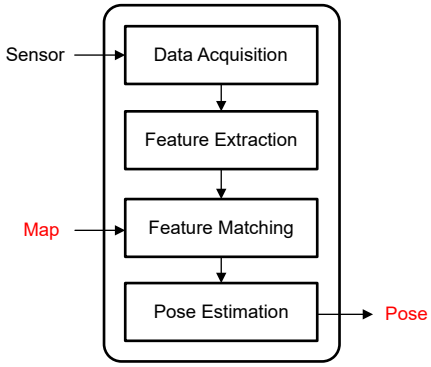

Figure 2-3. Localization Process With Pre-built Map

Figure 2-3 shows the localization process in the mapped environment. This process is similar to the SLAM front end in Figure 2-2. The only difference is that, once the features are extracted from a frame, the corresponding features are searched from the map instead of ones from other frames. After finding the matched features, the pose of the mobile robot can be calculated by Perspective-n-Point (PnP), Iterative Closest Point (ICP), and so forth.

Table 2-1 summarizes the widely used technique in each step of the graph SLAM and localization.

Table 2-1. Various Techniques in Graph SLAM

	Visual SLAM	LiDAR SLAM
Feature extraction	• Feature detection and descriptor: – SIFT, SURF, KAZE, AKAZE, ORB, BRISK – Convolutional Neural Network (CNN) based	• 2D or 3D points itself • Geometric features – Edge – Plane
Feature association	• Minimum Euclidean or Hamming distance between feature descriptors	• Minimum Euclidean distance between 2D or 3D points, edges, and planes
Pose estimation	• Direct Linear Transform (DLT) • LM optimization • PnP	• Scan matching – Occupancy grid matching – ICP – Normal Distribution Transform (NDT) • LDT, LM optimization
Loop closure detection	• Bag of Word (BoW) • Appearance-based global image descriptor • CNN based	• Scan matching • Segmentation matching • CNN based
Graph optimization (Bundle adjustment)	• Make use of matrix sparsity • Newton, Gauss-Newton, LM underneath	• Make use of matrix sparsity • Newton, Gauss-Newton, LM underneath

The AM69A embedded processor is an excellent choice for SLAM and localization. Octal A72 cores provide more than enough computing power for complex SLAM and localization algorithms. Many open-source algorithms can be quickly implemented and benchmarked on the AM69A. Moreover, the functional blocks such as feature extraction, feature matching, and pose estimation in Figure 2-2 and Figure 2-3 can be offloaded to hardware accelerators (HWAs) and C7x DSP to improve performance. Internal studies show that the throughput of the ORB SLAM with stereo camera is improved by 2 to 3 times by offloading stereo rectification and feature extraction to LDC, MSC, and DSP.

3 Surroundings Perception

The mobile robot must comprehend the dynamic change of the surroundings for safe navigation. Dynamic obstacles getting in the way of the mobile robot must be detected and avoided as quickly as possible. The mobile robot can detect obstacles in the middle of localization as well since the features from obstacles do not exist in the map. Identifying these features is not always successful though. Sometimes the features belonging to obstacles are wrongly matched to the features in the map. Therefore, it is important to identify and exclude the features belonging to obstacles for reliable localization.

This is where AI and sensor fusion with other sensor modalities such as millimeter Wave (mmWave) radar come into the picture. Both AI and sensor fusion help the mobile robot perceive dynamic objects accurately and therefore help navigate more safely and intelligently. Vision-based deep learning networks are able to detect and classify obstacles and measure their distances to the mobile robot. TI mmWave radar is unique in the sense that the radar provides range, velocity, and angle-of-arrival information of obstacles, with which the robot can navigate better without collision.

The AM69A processor has four 512-bit C7x DSPs running at 1 GHz, each of which is tightly coupled with each of four MMAs capable of 4K 8-bit fixed multiply-accumulate (MAC) per cycle. Four MMAs provides 32 dense Trillion Operations per Second (TOPS) that can support various deep-learning networks with multiple sensors simultaneously. In addition, the AI application development on AM69A is made simpler and faster with the Processor SDK Linux for AM69A. This Software Development Kit (SDK) enables an interplay of multiple open-source components and deep-learning runtime such as TFLite, ONNX, and TVM on top of the foundational Linux® component and the firmware packages for remote cores and HWAs. TI has converted and exported 100+ models from their original training frameworks in PyTorch, TensorFlow, and MXNet into the format friendly to the

C7xMMA architecture and hosts them in Edge AI Model Zoo. TI also provides Edge AI Studio that is a collection of tools to accelerate the development of edge AI applications on TI's embedded processors including AM69A. Edge AI Studio allows building, evaluation and deployment of deep learning models. The TI E2E™ forums article *How to simplify your embedded edge AI application development* has more about free tools and software from TI designed to help with the development process.

4 Path Planning

With localization and surroundings perception, the mobile robot has all the necessary information to determine a path and navigate safely to the destination. A wide variety of path-planning algorithms are available. Among them, the path planning on the occupancy grid map is simple yet effective. Moreover, since obstacles are detected while updating the occupancy grid map, the occupancy grid map is widely used for obstacle detection and path planning. By identifying occupied and unoccupied grids, the mobile robot can discover an efficient and safe path to the destination.

Figure 4-1 is an example data flow of the autonomous navigation system on the AM69A using an image sensor mounted on the mobile robot. The raw image is processed and demosaiced by VPAC3 VISS and the image distortion is removed by LDC. For surroundings perception, the undistorted image is re-sized by MSC and converted to RGB format that DL networks operate on. DL inferencings are accelerated by MMA to detect objects and determine their poses from the image sensor. For localization, the feature points are extracted from the undistorted image and the corresponding feature points are searched from the map to determine the pose of the mobile robot on the map. The C7x DSP can optimize this localization process. Many feature extraction algorithms use pyramid image and MSC can accelerate the pyramid image generation. The pose of the mobile robot and the detected objects are input to path planning. Since the detected objects can be projected onto the map as well using the pose of the mobile robot, the free spaces around the mobile robot are identified and therefore the path to the destination and the control command for the mobile robot are determined. This example data flow can be extended with multiple image sensors with proper extrinsic calibration between sensors.

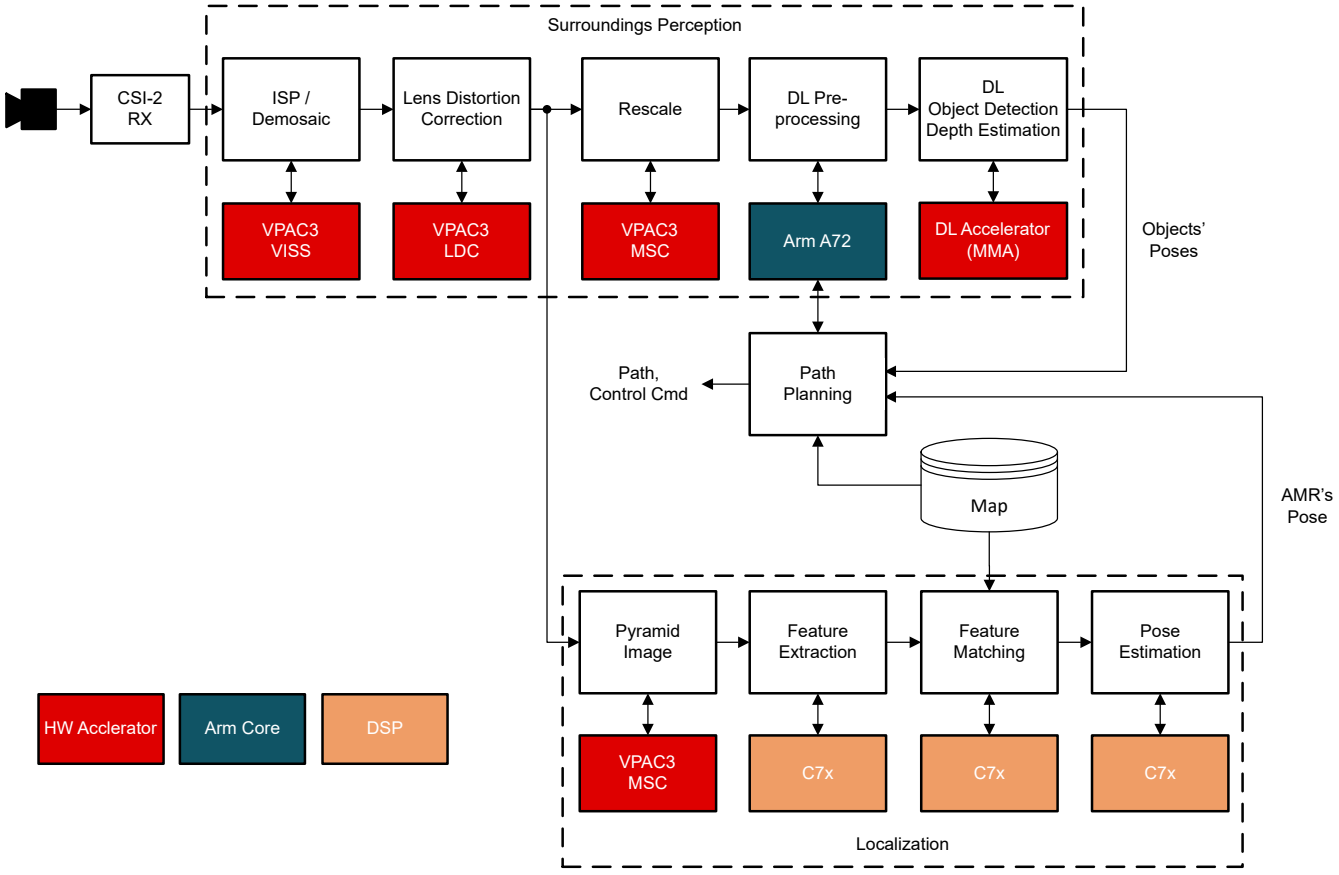

Figure 4-1. Example Data Flow of AM69A Autonomous Navigation System

The autonomous navigation system using 2D LiDAR and IMU was implemented on top of the Robotics SDK and featured with the SCUTTLE robot. For this system, the occupancy grid map is built offline using the 2D LiDAR SLAM and the map is used for real-time navigation using the same 2D LiDAR sensor. For autonomous navigation, the three tasks in Figure 1-1, that is, localization, surroundings perceptions, and path planning are performed with every LiDAR scan and the control command is converted to a Pulse Width Modulation (PWM) signal to control motors. The data flow is described in the project page and this system can be duplicated on the AM69A processor.

5 Summary

This white paper provides the technical overview on mapping, localization, dynamic environment perception, and path planning, which are the central pieces of the autonomous navigation of mobile robots. Adding AI makes the robots more intelligent and safer, and therefore improves productivity and efficiency in broader applications including home automation, factory automation, warehousing, and logistics. The AM69A processor helps build a well-designed platform for the autonomous mobile robots with the significant levels of computing power, AI performance, and easy-to-use software development tools and support.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265